HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr Ing.
Christian Neuhold

Datendeduplizierung auf Basis freier Software in einer Enterprise-Virtualisierungsumgebung

Mittweida, 2012

Fakultät Elektro- und Informationstechnik

DIPLOMARBEIT

Datendeduplizierung auf Basis freier Software in einer Enterprise-Virtualisierungsumgebung

Autor:
Herr Ing.

Christian Neuhold

Studiengang:
Informationstechnik

Seminargruppe:
KI09SHA

Erstprüfer:
Prof. Dr.-Ing. habil. Lutz Winkler

Zweitprüfer:
Dipl.-Ing. (FH) Harald Leopold

Einreichung:
Mittweida, 02.11.2012

Verteidigung/Bewertung:
Mittweida, 2012

Bibliografische Angaben:

Neuhold, Christian:
Datendeduplizierung auf Basis freier Software in einer Enterprise-Virtualisierungsumgebung - 2012 - 11, 64, 17 S.
Mittweida, Hochschule Mittweida (FH), University of Applied Sciences, Fakultät Elektro- und Informationstechnik, Diplomarbeit, 2012

Referat:

Das Eindämmen des Datenwachstums stellt für Unternehmen im IT-Outsourcing-Bereich eine große Herausforderung dar. In dieser Arbeit wird untersucht, ob eine Integration eines DEDUP-Systems auf Basis freier Software in einer ESX-Umgebung möglich ist.
Hierfür werden die Projekte SDFS und LessFS als freie Softwareprojekte im Datendeduplizierungsumfeld vorgestellt. Durch definierte Auswahlkriterien und eine anschließende ESX-Laborumgebung werden die Potenziale und Schwächen eines DEDUP-Systems aufgezeigt.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Interface
BHS	Basic Header Segment
CIFS	Common Internet Filesystem
CDB	Command Description Block
DNS	Domain Name System
DRE	Data Redundancy Elimination
DSE	Deduplication Storage Engine
DEDUP	Deduplizierung
FC	Fibre Channel
FUSE	Filesystem in Userspace
ISBN	Internationale Standardbuchnummer
ISCSI	Internet Small Computer System Interface
ISO	International Standards Organization
IETF	Internet Engineering Task Force
ISID	Initiator Session Identifier
LUN	Logical Unit Number
SSID	Session ID
SCSI	Small Computer System Interface
MAN	Metropolitan Area Network
NFS	Network File System
NTP	Network Time Protocol
OVF	Open Virtualization Format
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
POSIX	Portable Operating System Interface
RAID	Redundant Array of Independent Disks
RFC	Request for Comments
RPC	Remote Procedure Call
SDFS	Single Deduplication File system
SIS	Single Instance Store
SNIA	Storage Networking Industry Association
SMB	Server Message Block
TSID	Target Session Identifier

VAAI	vStorage APIS for Array Integration
WAN	Wide Area Network
XDR	External Data Representation
XML	Extensible Markup Language

1 Übersicht

1.1 Motivation

Virtuelle Umgebungen ersetzen in den heutigen Rechenzentren immer mehr die klassischen Serversysteme. Wurde früher versucht, IT-Dienstleistungen auf wenige, aber dafür speziell ausgerüstete Serversysteme zu konsolidieren, wird in einer virtuellen Umgebung für eine IT-Dienstleistung oft eine dedizierte virtuelle Maschine eingesetzt.

Diese Strategie garantiert einem Betreiber eines IT-Outsourcing-Unternehmens maximale Flexibilität für die angebotenen IT-Dienstleistungen. Kundenwünsche und Forderungen können schnell und effizient umgesetzt werden, da es ein Leichtes ist, über die Virtualisierungsschicht mehr oder weniger Ressourcen für eine IT-Dienstleistung bereitzustellen.

Nachteilig wirkt sich der Verbrauch bei den Storage-Systemen aus. Im Gegensatz zu der Netzwerkanbindung, CPU-Zeit oder dem Hauptspeicher können Storage-Systeme nicht dynamisch angepasst werden. Der Festplattenspeicher ist fest mit der virtuellen Maschine verbunden. Jede neue Maschineninstanz benötigt zusätzlichen Festplattenspeicher für das Betriebssystem und deren Applikationen.

Diese Art der Bereitstellung von IT-Dienstleistungen schlägt sich auf die Betriebskosten nieder, die sich nicht nur aus den Kosten des Festplattenspeichers, sondern auch aus jenen des Stromverbrauchs, der benötigten Klimatisierung und des Platzverbrauchs in einem Serverraum zusammensetzen. Technologien, um dieses Datenwachstum und damit die Kosten einzudämmen, sind bei einem IT-Outsourcing-Unternehmen gefragt.

1.2 Zielsetzung

Virtuelle Umgebungen, wie zum Beispiel die Virtualisierungssoftware ESX von der Firma VMware, unterstützen nur einen Teil der vorhandenen Betriebssystemarten am Softwaremarkt. Aktuell ermöglicht ESX Betriebssysteme zu virtualisieren, die auf der x86 oder x86_64-Architektur basieren. Andere Architekturen wie zum Beispiel die Architektur SPARC von der Firma Oracle oder PowerPC von der Firma IBM sind nicht möglich. Diese Einschränkung führt dazu, dass sich Rechenzentren von einer heterogenen zu einer immer mehr homogenen Umgebung entwickeln.

Auf den zentralen Storage-Systemen, die meist als Backend für die Virtualisierungsumgebung eingesetzt werden, befinden sich immer öfters virtuelle Maschinen, die mit dem gleichen oder einem ähnlichen Betriebssystem arbeiten. Die virtuellen Maschinen unterscheiden sich deutlich in den erbrachten IT-Dienstleistungen, die Unterschiede auf der Betriebssystemebene sind gering. Dieser geringe Unterschied führt schlussendlich dazu, dass ein Storage-System zum Großteil idente Daten verarbeitet. Die Effizienz, aus der Sicht des Speicherplatzes, sinkt mit jeder neuen virtuellen Maschine. Die zentralisierte

Speicherung bietet aber auch Vorteile, die früher mit einzelnen dedizierten Servern nicht möglich waren.

Um den Verbrauch in den Storage-Systemen zu reduzieren und die Nutzung effizienter zu gestalten, werden Systeme eingesetzt, die Daten von Redundanzen befreien. Vermehrt wird auf Software-Appliances[1] oder Hardware-Appliances[2] gesetzt, die solche Funktionalitäten abbilden. Allgemein werden sie als Deduplizierungssysteme oder kurz DEDUP-Systeme bezeichnet.

Diese Diplomarbeit beschäftigt sich mit DEDUP auf Basis von freier Software. Es soll geklärt werden, ob es möglich ist, mit freier Software eine DEDUP-Lösung in einer Enterprise-Virtualisierungsumgebung zu integrieren. Hauptaugenmerk wird auf die technische Durchführbarkeit gelegt.

1.3 Kapitelübersicht

Diese Diplomarbeit besteht aus insgesamt fünf Kapiteln. **Kapitel 1** beginnt mit einer Übersicht der Diplomarbeit, gefolgt vom Abschnitt Motivation und der Zielsetzung.

In **Kapitel 2** wird allgemein der Begriff DEDUP erläutert und der historische Hintergrund beleuchtet. Darauf aufbauend werden die unterschiedlichen Arten der DEDUP-Technologien vorgestellt und genauer auf die blockbasierende Deduplizierung eingegangen. Es werden die verschiedenen Arten und deren Einsatzmöglichkeit beschrieben. Abschließend erfolgt eine nähere Betrachtung der benötigten Datenübertragungsprotokolle.

Die Anforderung an eine DEDUP-Lösung in einer virtuellen Serverumgebung und die darauf folgende Auswahl eines geeigneten Softwareprojekts sind die Themen in **Kapitel 3**. Dieses Kapitel beginnt mit der Definition der Arbeitspakete, die im vierten Kapitel in einer Praxisumgebung untersucht werden. Abschließend wird eine Übersicht der am freien Softwaremarkt existierenden Lösungen für die Datendeduplizierung gegeben und begründet, welches Softwareprojekt bei den praktischen Untersuchungen ausgewählt wird.

Der praktische Einsatz einer Datendeduplizierung in einer Virtualisierungsumgebung wird in **Kapitel 4** beschrieben und protokolliert. Anhand einer VMware-ESX-Virtualisierungsumgebung und mehrerer virtueller Maschinen wird die Datendeduplizierungssoftware überprüft. Es werden die technischen Möglichkeiten, die Grenzen und Performance der Open-Source-DEDUP-Lösung untersucht.

In **Kapitel 5** werden die Ergebnisse zusammengefasst und ein Ausblick auf die Möglichkeiten der Arbeit mit einer freien Datendeduplizierung gegeben.

---

[1] Bei einer Software-Appliance handelt es sich um eine vorkonfigurierte virtuelle Maschine, die für eine bestimmte Aufgabe entwickelt wurde.

[2] Die Hardware-Appliance erfüllt wie die Software-Appliance eine bestimmte Aufgabe, kombiniert diese mit einer geeigneten Hardware.

2 Grundlagen

Unter dem Begriff DEDUP werden viele unterschiedliche Technologien und Produkte am aktuellen Softwaremarkt zusammengefasst. Angefangen bei der Datenkompression bis zu den sogenannten WAN-Beschleunigern. Im Kapitel Grundlagen werden häufig verwendete Begriffe im DEDUP-Umfeld erklärt und die Kernelemente einer DEDUP-Lösung auf Blockbasis beschrieben.

2.1 Begriff Datendeduplizierung

Die Bedeutung des Begriffs Datendeduplizierung hat sich in den letzten Jahren stark verändert. Bei den ersten Implementierungen dieser Technologie wurde hierfür der Begriff Single-Instance-Store-Technologie (SIS[3]-Technologie) verwendet. In der aktuellen Form wird er genutzt, um DEDUP auf Basis einer Block-Datendeduplizierung zu beschreiben.

Die Organisation SNIA[4] definiert den Begriff DEDUP im Bereich Storage-Systeme wie folgt

"[Storage System] The replacement of multiple copies of data – at variable level of granularity – with references to a shared copy in order to save storage space and/or bandwidth" [SNIA2012]

Diese allgemeine Definition schließt alle aktuell am Markt erhältlichen DEDUP-Systeme mit ein. Sobald redundante Daten durch ein System gelöscht und durch eine Referenz auf die einzigartigen Daten ersetzt werden, handelt es sich um eine Datendeduplizierung. Dabei ist es nicht auschlaggebend, ob dieser Prozess bei der Datenübertragung oder beim Speichern von Daten durchgeführt wird.

DEDUP-Systeme im Datenspeicherungsumfeld sind in der Funktionsweise ähnlich, unterscheiden sich jedoch in der Art der Verarbeitung, wie zum Beispiel in der Duplikats-Erkennung oder in der Granularität der verarbeiteten Daten.

Die nachfolgende Abbildung 1 zeigt den logischen Ablauf einer Datendeduplizierung bei einer Datenspeicherung. Die Grundbausteine sind bei diesen Systemen immer dieselben. Daten werden mittels eines Duplikats- Erkennungsalgorithmus analysiert und in Unikate und Duplikate unterteilt. Handelt es sich um Unikate, werden sie gespeichert, Duplikate

[3] SIS Single Instance Store ist eine Technologie, die Redundanzen auf Dateiebene entfernt (siehe Abschnitt 2.4.1).

[4] SNIA (Storage Networking Industry Association) ist eine Non-Profit-Handelsvereinigung mit dem Schwerpunkt Daten und Speichernetzwerke (siehe http://www.snia.org).

werden durch sogenannte Referenzen auf die entsprechenden Unikate ersetzt. Die Daten vom Duplikat werden schlussendlich verworfen.

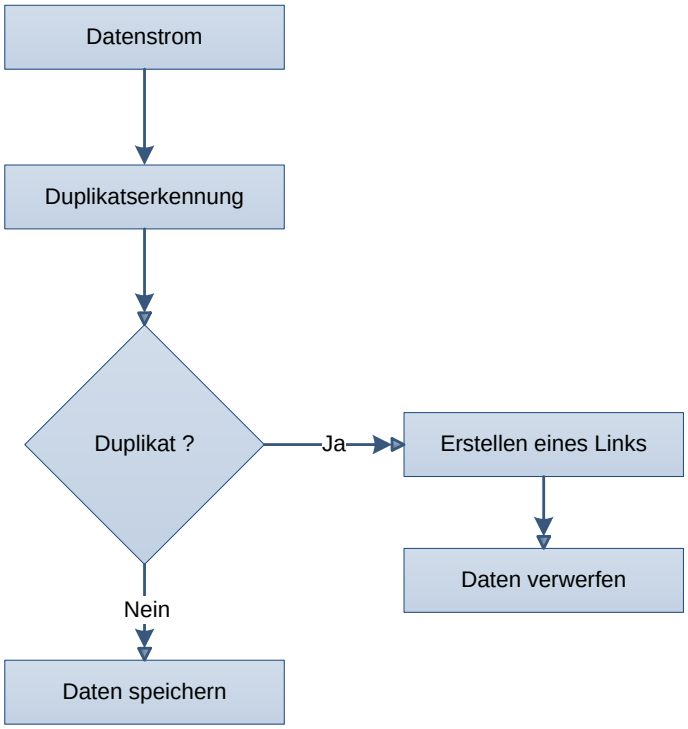


Abbildung 1: DEDUP-Verlauf

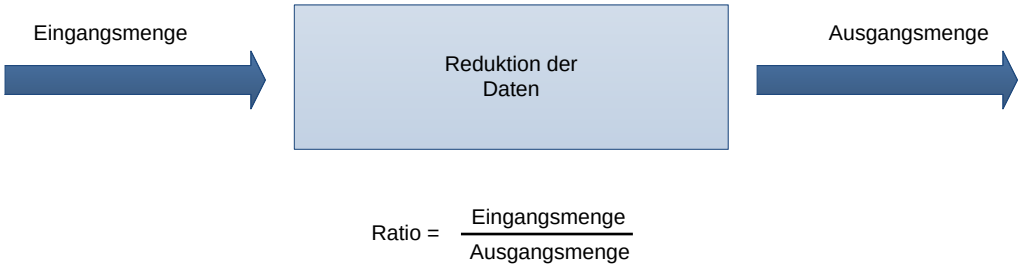
Die Speicherplatzeinsparung wird durch das Ersetzen von Duplikaten durch eine Referenz erreicht. Eine Referenz benötigt nur einen Bruchteil des Speicherplatzes gegenüber jenem des Duplikats. Auch wenn die für die Verwaltung der Referenzen benötigte Datenmenge hinzugerechnet wird, überwiegt die Einsparung des Speicherplatzes.

In einem DEDUP-System ist die zuverlässige Erkennung von Duplikaten die wichtigste Eigenschaft. In diesem Prozess werden die Geschwindigkeit, Genauigkeit und die Datensicherheit eines DEDUP-Systems bestimmt. Von den Herstellern werden die unterschiedlichsten Technologien eingesetzt, wobei das Hauptunterscheidungsmerkmal in der Granularität der verarbeiteten Daten liegt.

2.2 DEDUP-Ratio

Laut [SNI2008] legt der Begriff DEDUP-Ratio fest, inwieweit Daten durch ein DEDUP-System reduziert werden können. Hierbei werden über eine bestimmte Zeitperiode die eingehenden Daten mit den ausgehenden in Verhältnis gesetzt.

In der Abbildung 2 wird DEDUP-Ratio-Formel dargestellt.



$$\text{Ratio} = \frac{\text{Eingangsmenge}}{\text{Ausgangsmenge}}$$

Abbildung 2: DEDUP-Ratio

Angegeben wird der Wert meist in der Form „X:ratio“, wobei X den Ratio-Wert darstellt. Ein DEDUP-System, welches einen Ratio-Wert von 10:1 erreicht, kann 100 GB an Daten auf 10 GB reduzieren. Viele Hersteller geben bei ihren DEDUP-Systemen die wahrscheinliche Speichermenge an und verschleiern damit die tatsächliche Speicherkapazität der Systeme.

2.3 Historie

Bei der Datenkompression handelt es sich um eine der ersten Technologien, um Daten zu reduzieren. Sie ermöglicht, wiederholende Bereiche zu finden und diese neu zu kodieren. Bei dieser Neukodierung wird eine günstigere Repräsentation gesucht, die es erlaubt, die originale Information in einer kürzeren Form darzustellen. Für die Suche und neue Kodierung dieser Bereiche werden die unterschiedlichsten Algorithmen eingesetzt, die sich durch verlustbehaftete oder verlustfreie Kodierung unterscheiden.

Diese Technologien besitzen meist die Limitierung, nur einzelne Dateien auf der Dateiebene oder Datenströme in einer Datenkommunikation zu verarbeiten. Um eine hohe Datenreduktion bei einem Storage-System zu erhalten, wird eine Datenkompression über mehrere Dateien hinweg benötigt. Mit der klassischen Datenkompression ist dies nur mit erheblichem Aufwand möglich. [INF2006]

Aufbauend auf dieser Technologie wurde versucht diese Einschränkung aufzuheben. Die dafür entwickelte SIS-Technologie hob die Dateigrenzen auf und ermöglichte es erstmals, Dateien miteinander zu vergleichen. Diese Technik wurde vor allem bei E-Mail-Systemen und Archivierungssystemen eingesetzt. Durch eine nachgelagerte Datenkompression konnte die Effizienz weiter gesteigert werden. In Abschnitt 2.5.1 wird auf SIS näher eingegangen.

Datendeduplizierung auf Blockebene ist die aktuellste Technologie, um Duplikate zu erkennen und zu eliminieren. Die Erkennung von Duplikaten wird nicht innerhalb einer Datei oder auf Dateiebene durchgeführt, sondern in Blöcken. Abbildung 3 zeigt beispielhaft die Deduplizierung auf Blockebene.

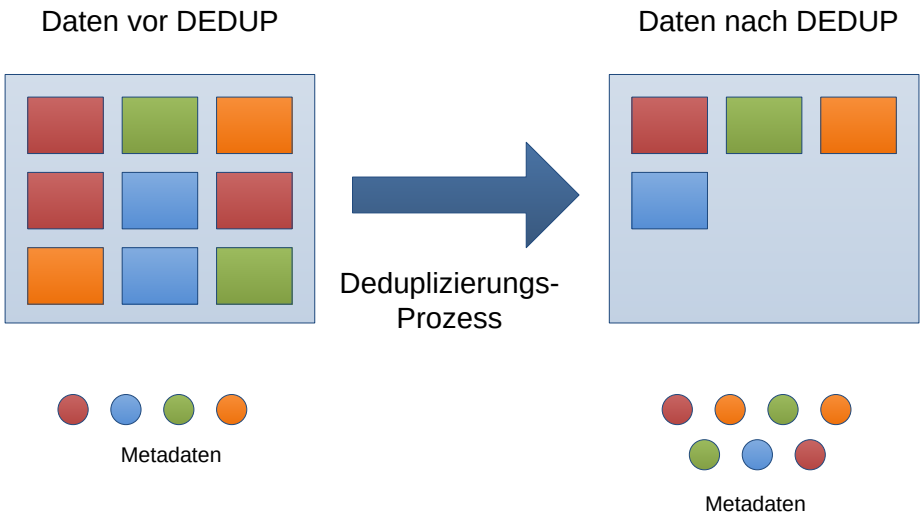


Abbildung 3: Block-DEDUP

Erfolgt eine Datenspeicherung auf ein blockbasierendes DEDUP-Speichersystem, werden diese Daten in Blöcke zerlegt, verglichen, dedupliziert und erst danach auf das Speichermedium geschrieben. Das Entfernen der redundanten Daten und Wiederherstellen der Originaldaten erfolgt meist transparent gegenüber dem Betriebssystem und erfordert keine zusätzlichen Treiber oder Programme. Nach einer Deduplizierung wird für die Verwaltung der Unikate mehr Speicherplatz benötigt. Dadurch steigt der Platzbedarf der Metadaten im Vergleich zu einem Speichersystem ohne eine DEDUP-Funktion an.

2.4 Einsatzgebiete

DEDUP ist in den verschiedensten Bereichen der Informationstechnik präsent. Hier soll ein Überblick zu deren häufigen Anwendungsgebieten gegeben werden.

2.4.1 Datensicherung

Wie in [Data2009] beschrieben, setzen klassische Datensicherungssysteme auf inkrementelle, differentielle und Vollsicherungen. Diese Basistechnologien werden miteinander kombiniert, um einen Kompromiss zwischen der Dauer einer Sicherung, Rücksicherung und dem Speicherplatzverbrauch zu finden.

Oftmals stoßen diese Technologien an ihre Grenzen, wenn es erforderlich ist, Daten über größere Übertragungsstrecken zu transportieren. Die Datenraten sind über längere Distanzen meist begrenzt und die Übertragung der Datensicherung ist in der erforderlichen Zeit nicht mehr möglich.

Der Einsatz von DEDUP in einer Datensicherungssoftware ermöglicht es, die gegebenen Ressourcen besser zu nutzen. Datensicherungsprodukte, wie zum Beispiel das Produkt Tivoli Storage Manager von IBM, das auf eine Server/Client-Architektur setzt, bieten meist zwei Möglichkeiten der DEDUP-Implementierung an, die nachfolgend genauer betrachtet werden.

2.4.1.1 DEDUP am Sicherungsclient [IBMC2012]

Wird das Entfernen von redundanten Daten direkt am Sicherungsclient durchgeführt, ist die Infrastruktur zwischen dem Sicherungsserver und dem Sicherungsclient entlastet. Anstatt immer wieder die gesamten Daten zu übertragen, reicht es, die einzigartigen Daten an den Sicherungsserver zu schicken, um ein vollständiges Backup zu erhalten. Im Grunde wird die inkrementelle Sicherung, die auf Dateiebene arbeitet, auf die Blockebene erweitert.

Nachteile birgt die Datendeduplizierung durch den höheren Ressourcenverbrauch auf dem Sicherungsclient und den erhöhten Aufwand für die Verwaltung. Jeder Sicherungsclient muss mit einem speziellen Client ausgestattet werden, der die DEDUP-Funktion beinhaltet. Dieser Client benötigt regelmäßige Wartung und während der Sicherung mehr Ressourcen für den zusätzlichen DEDUP-Prozess als eine Sicherungsclient, der diese Funktionalität nicht anbietet.

Abbildung 4 zeigt die Datendeduplizierung am Sicherungsclient. Zusätzlich zum Datenstrom, der ausschließlich aus Unikaten besteht, wird eine zweite Verbindung benötigt, die die Hashwerte überträgt. Diese Hashwerte dienen dazu, um festzustellen, ob es sich um Unikate oder Duplikate handelt. Vor der Übertragung der Unikate zum Sicherungsserver kann eine zusätzliche Datenkompression eingesetzt werden, um auch die Größe der Unikate zu verkleinern.

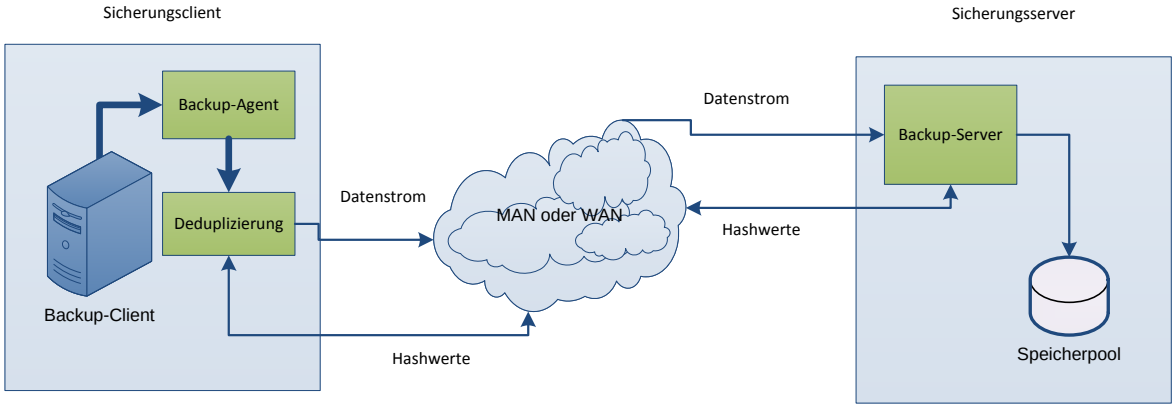


Abbildung 4: DEDUP am Sicherungsclient

Obwohl sehr viele Punkte gegen eine Datendeduplizierung am Client sprechen, existieren Einsatzgebiete für diese Technologie. Das häufigste Beispiel ist eine Datensicherung, die über eine Wide-Area-Network-Strecke (WAN-Strecke) oder Metropolitan-Area-Network-Strecke (MAN-Strecke) übertragen wird. Durch die Reduzierung der Daten vor der Übertragung an den Sicherungsserver ist eine Datensicherung über geografisch weit entfernte Orte möglich.

2.4.1.2 DEDUP am Sicherungsserver [IBMS2012]

Die zweite Möglichkeit ist, die Deduplizierung am Sicherungsserver durchzuführen. Diese Methode entlastet nicht die Infrastruktur zwischen dem Client und dem Server wie bei der

Client-Deduplizierung, sondern gezielt den Speicherpool am Sicherungsserver. Daten werden wie gewohnt inkrementell, differentiell oder durch eine Vollsicherung wie bei einer herkömmlichen Datensicherung übertragen. Nach der Übertragung übernimmt der Sicherungsserver den Prozess der Erkennung und Entfernung redundanter Daten.

Hierbei werden die Clients nicht zusätzlich mit der Deduplizierung der Daten belastet und müssen weniger Ressourcen für die Datensicherung bereitstellen. In der Abbildung 5 ist der Datenstrom zwischen Sicherungsclient und Sicherungsserver dargestellt. Die Strichstärke symbolisiert die Menge an Daten, die übertragen wird. Ab der Deduplizierung reduziert sich diese auf die Unikate.

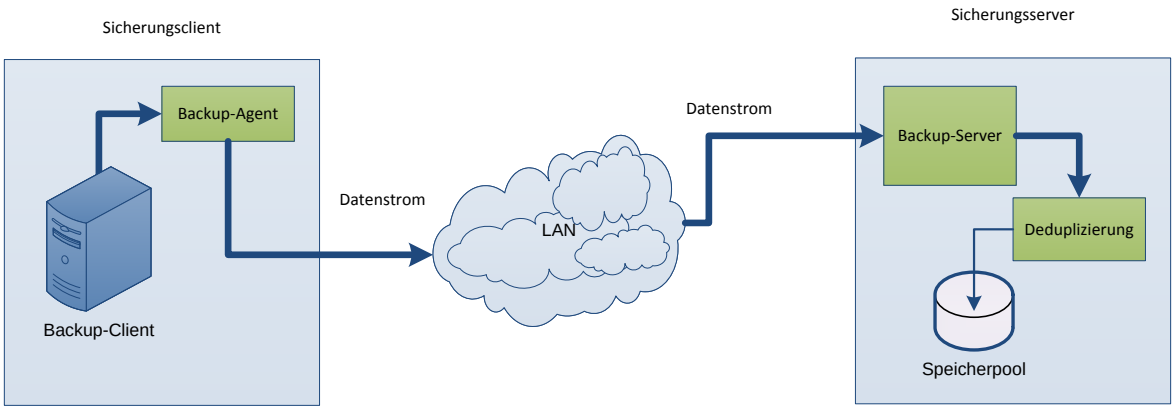


Abbildung 5: DEDUP – Sicherungsserver

Die höhere Leistungsanforderung an den Sicherungsserver muss bei der serverseitigen Deduplizierung berücksichtigt werden. Im schlimmsten Fall ist es dem Sicherungsserver nicht mehr möglich, in der gewünschten Zeit die Datensicherungen durchzuführen, da der Prozess der Deduplizierung zu hohe Ansprüche an die gegebenen Ressourcen stellt.

2.4.2 Datenübertragung

Die Entfernung von redundanten Daten findet auch in der Datenübertragung Anwendung. In WAN- und MAN-Netzwerken ist aus Kostengründen oft die Datenrate begrenzt. Dadurch ist es umso wichtiger, die vorhandenen Kapazitäten effektiv auszunutzen. Die DRE[5]-Technologie, die in den Cisco-WAAS-Produkten eingesetzt wird, kann in solchen Konfigurationen die Datenübertragung optimieren. [CIS2011]

DRE-Technologie ermöglicht es, wiederkehrende Bytesequenzen in einer Datenübertragung zu erkennen und zu ersetzen. Es werden dazu wiederholende Sequenzen in einer Datenbank gespeichert und mit einem Identifikationscode versehen. Wird bei einer Datenübertragung eine dieser definierten Sequenzen erkannt, werden die Daten durch den entsprechenden Identifikationscode, der in einer Datenbank gespeichert ist, ersetzt. Anstatt der Originaldaten wird der Datenstrom mit dem Identifikationscode neu kodiert.

[5] DRE – Data Redundancy Elimination eliminiert wiederkehrende Datenfolgen.

Die Gegenstelle erkennt die Identifikationscodes und ersetzt diese wieder durch die ursprünglichen Daten. Dahinterliegende Systeme müssen nicht angepasst werden, da das Ersetzen und Wiedereinfügen der Daten komplett transparent erfolgt. Dadurch, dass die Identifikationscodes um ein Vielfaches kleiner als die zu übertragenden Daten sind, kommt es zu einer Optimierung der Datenübertragung. Nicht nur die Geschwindigkeit, sondern auch die Zugriffszeiten werden verkürzt. Abbildung 6 zeigt den Aufbau einer WAN-Strecke mit der DRE-Technologie.

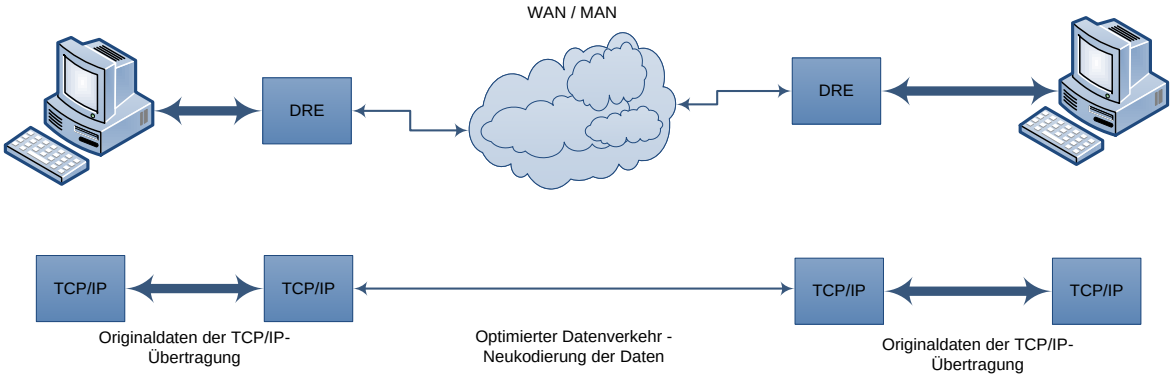


Abbildung 6: DRE-TCP/IP-Optimierung

2.4.3 Storage-Systeme

DEDUP wird vermehrt in festplattenbasierenden Storage-Systemen eingesetzt. Diese Funktionalität erhöht die Speicherkapazität um ein Vielfaches. Hersteller geben eine bis zu 20-fache Reduktion der Daten an. Das ist gleichbedeutend mit einer 20-fach höheren Speicherkapazität, als ein System ohne DEDUP hat. Im Praxisumfeld laut [SPE2010] wird eine Datenreduktion von typischerweise 12:1 erreicht.

Diese Datenreduktion ermöglicht es, Bandlaufwerke oder Bandroboter durch Festplattensysteme zu ersetzen. Speziell wenn eine hohe Datensicherungsgeschwindigkeit und vor allem Rücksicherungsgeschwindigkeit gefordert wird, ist dies sinnvoll. Die Zeit für das Suchen und Einlegen der Bänder entfällt und eine Rücksicherung ist durch die schnellen Zugriffszeiten der Festplatten sofort möglich.

Eine Einteilung der Daten und Speichersysteme in Tier-Klassen ist eine weitere Variante, um DEDUP-Systeme einzusetzen. Die Einteilung erlaubt es, Daten mit einer hohen Anforderung an Geschwindigkeit und Zugriffzeit auf einem leistungsstarken Storage-System abzulegen, Daten mit geringeren Anforderungen auf einem leistungsschwächeren Storage-System.

Storage-Systeme, die im unteren Bereich der Tier-Klassen eingesetzt werden, sind auf Kapazität und nicht auf Performance ausgelegt – Anforderungen, die ein DEDUP-System sehr gut erfüllt. Im oberen Bereich werden Storage-Systeme mit einer hohen Leistungsfähigkeit eingesetzt. Eine Reduktion der Speicherkapazität bei diesen Systemen und somit auch der Kosten ist aufgrund der Tierklassen möglich. Im Gesamten betrachtet erhält man

ein hierarchisches Storage-System, das die Geschwindigkeit und auch die geforderte Kapazität kostengünstig liefern kann.

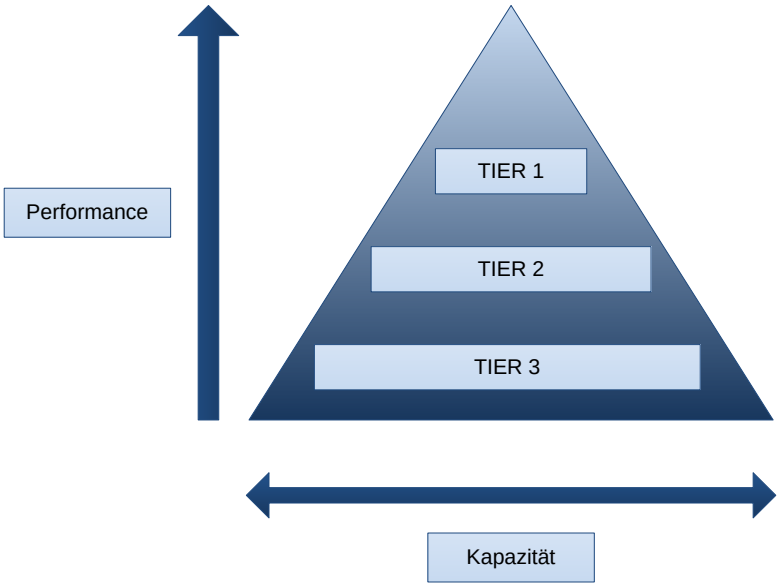


Abbildung 7: TIER – Storage-System

Abbildung 7 zeigt ein Storage-System, das in Schichten eingeteilt ist, und das Verhältnis von Kapazität zu Performance.

2.5 DEDUP-Technologien

Datendeduplizierung findet auf unterschiedlichste Arten und Ebenen in einer IT-Umgebung Einsatz. Die Granularität der verarbeiteten Daten hat einen direkten Einfluss auf die Performance eines DEDUP-Systems. Im folgenden Abschnitt wird eine Übersicht der DEDUP-Systeme und ihrer Technologien gegeben.

2.5.1 DEDUP auf Dateisystemebene

Bei der Daten-Deduplizierung auf Dateisystemebene werden redundante Dateien eliminiert. SIS hat sich als Begriff für diese Art der Deduplizierung etabliert.

Häufig wird SIS bei E-Mail-Archivierungssystemen eingesetzt. Dateianhänge, die bei einer E-Mail-Konversation innerhalb einer Firma ausgetauscht werden, sind oft ident. Benutzer A schickt ein E-Mail mit einem Dateianhang an Benutzer B innerhalb der gleichen Firma. Dieser Anhang wird ohne Einsatz von SIS zweimal gespeichert, einmal im „Posteingangsordner“ von Benutzer B und im „Gesendet-Ordner“ von Benutzer A. Mit der SIS-Technologie werden diese Anhänge verarbeitet und verglichen. Wenn es sich um die gleichen Anhänge handelt, wird einer davon gespeichert und der andere Anhang auf den einzigartigen Anhang referenziert.

Bei diesem vereinfachten Beispiel wird eine Datenreduzierung von 2:1 erreicht, wenn man den Speicherverbrauch der zusätzlichen Verwaltung für SIS vernachlässigt. Abbildung 8 zeigt den Unterschied zwischen einem System mit SIS und ohne SIS.

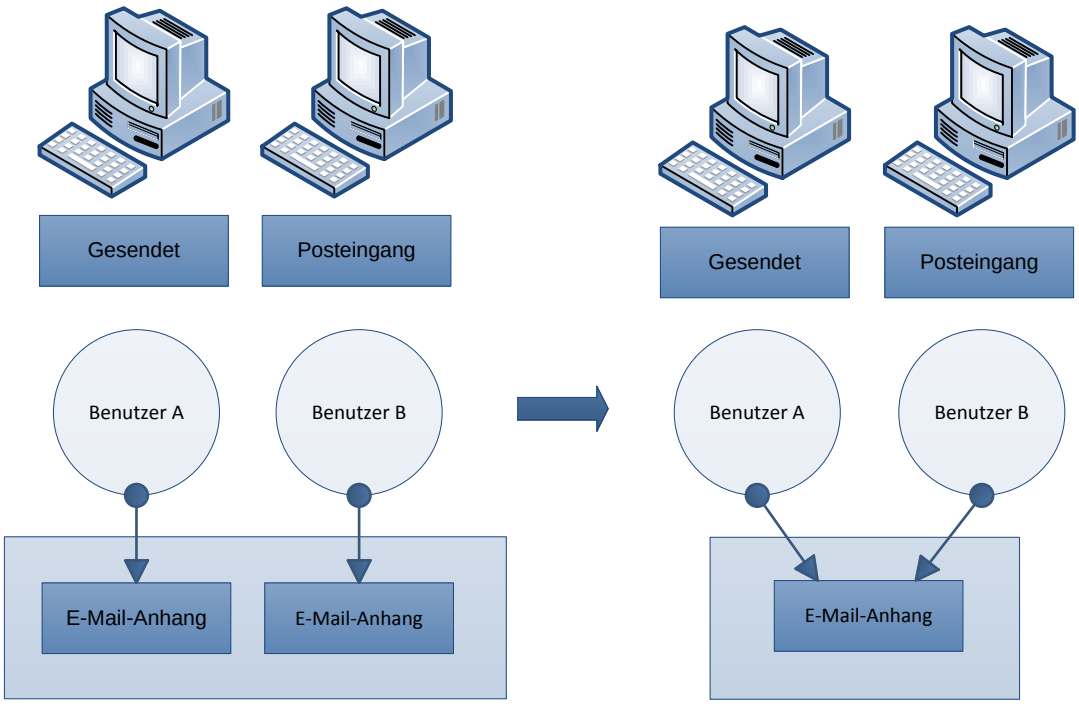


Abbildung 8: SIS-Technologie

Microsoft hat SIS in seinen Exchange[6]-Produkten ab Version 4.0 bis zu Version 2007 integriert. In der Version Exchange 2010 wurde diese Technik zugunsten von Kompressionsalgorithmen aufgegeben.

Die Effizienz der Datenreduktion sinkt erheblich, wenn nur eine minimale Änderung an den Dateien stattfindet. Dadurch, dass bei dieser Technologie die kleinste verarbeitete Einheit eine Datei ist, muss die komplette Datei erneut abgespeichert werden, wenn diese auch nur teilweise verändert wird.

2.5.2 DEDUP auf Blockebene

Datendeduplizierung auf Blockebene löst das Effizienzproblem der SIS-Technologie bei kleinen Veränderungen innerhalb einer Datei. Finden und Löschen gleicher Daten wird bei diesem Verfahren eine Schicht unter dem Dateisystem, dem Blocklevel, durchgeführt.

Anstatt ganze Dateien zu analysieren, werden bei diesem Verfahren die Daten in sogenannte Blöcke unterteilt. Ein Vergleich dieser Blöcke führt zu einem Duplikat oder Unikat. Wird eine Datei verändert, wird nicht wie beim SIS die komplette Datei noch einmal gespeichert, sondern nur die Blöcke, die von der Veränderung betroffen sind. Die Effizienz der Deduplizierung steigt um ein Vielfaches gegenüber dem SIS. [QUA2011]

In Abbildung 9 wird der Prozess der DEDUP auf Blockebene dargestellt. Zwei Dateien A und B werden von dem DEDUP-System verarbeitet. Im ersten Schritt findet die Blockein-

[6] Bei Microsoft Exchange handelt es sich um ein Mail- und Groupwaresystem vorwiegend in einer Microsoft-Windows-Umgebung.

teilung statt, anschließend erfolgt die Duplikats-Erkennung. Blöcke, bei denen es sich um Unikate handelt, werden gespeichert, die Duplikate auf diese referenziert.

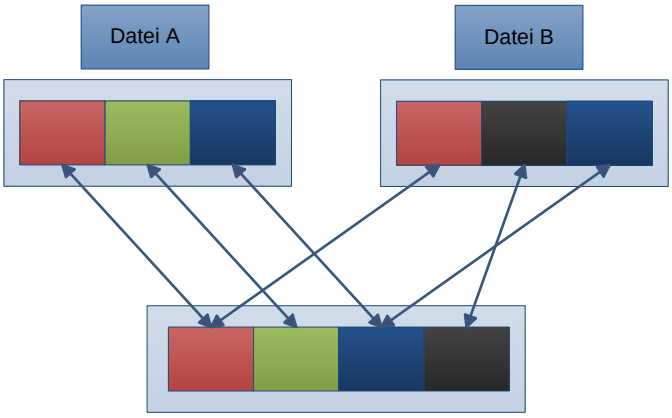


Abbildung 9: DEDUP-Blockebene

Bei der Erstellung der Blöcke wird zwischen zwei Arten unterschieden, der sogenannten konstanten oder variablen Blockgröße.

2.5.2.1 Konstante Blockgröße

Bei der konstanten Blockgröße wird die Einteilung der Daten in fest definierte Blöcke vorgenommen. Vorteile sind der geringere Ressourcenbedarf und die bessere Performance während der Verarbeitung gegenüber einer variablen Blockgröße, die im nächsten Abschnitt erläutert wird. Nachteilig ist die schlechtere DEDUP-Ratio. Dadurch, dass der DEDUP-Algorithmus sehr einfach ausgeführt ist, kann sich das System auf die unterschiedlichen Datentypen nicht einstellen und auf Veränderungen innerhalb einer Datei nicht optimal reagieren.

2.5.2.2 Variable Blockgröße

Die variable Blockgröße ist eine Weiterentwicklung der Variante mit konstanten Blockgrößen. Ein erweiterter Algorithmus zur Erkennung von Duplikaten passt die Blockgröße je nach Art der Daten an und kann dadurch die Effizienz der Datenreduktion steigern.

Ein typisches Beispiel für die Stärken der variablen Blockgrößen ist das Einfügen von neuen Daten am Anfang einer Datei. Bei einem System, das konstante Größen bei den Blöcken verwendet, wird jeder nachfolgende Block zu einem neuen einzigartigen Block, wenn die eingefügten Daten kleiner oder größer sind als die definierte Blockgröße.

In Abbildung 10 wird diese Situation dargestellt. Ein System mit variablen Blöcken erkennt die eingefügten Daten am Anfang des Datenstroms und versucht durch Anpassung der Blöcke, die dahinterliegenden Blöcke nicht zu beeinflussen. Dadurch entstehen weniger einzigartige Blöcke (Unikate) als bei den konstanten Blockgrößen.

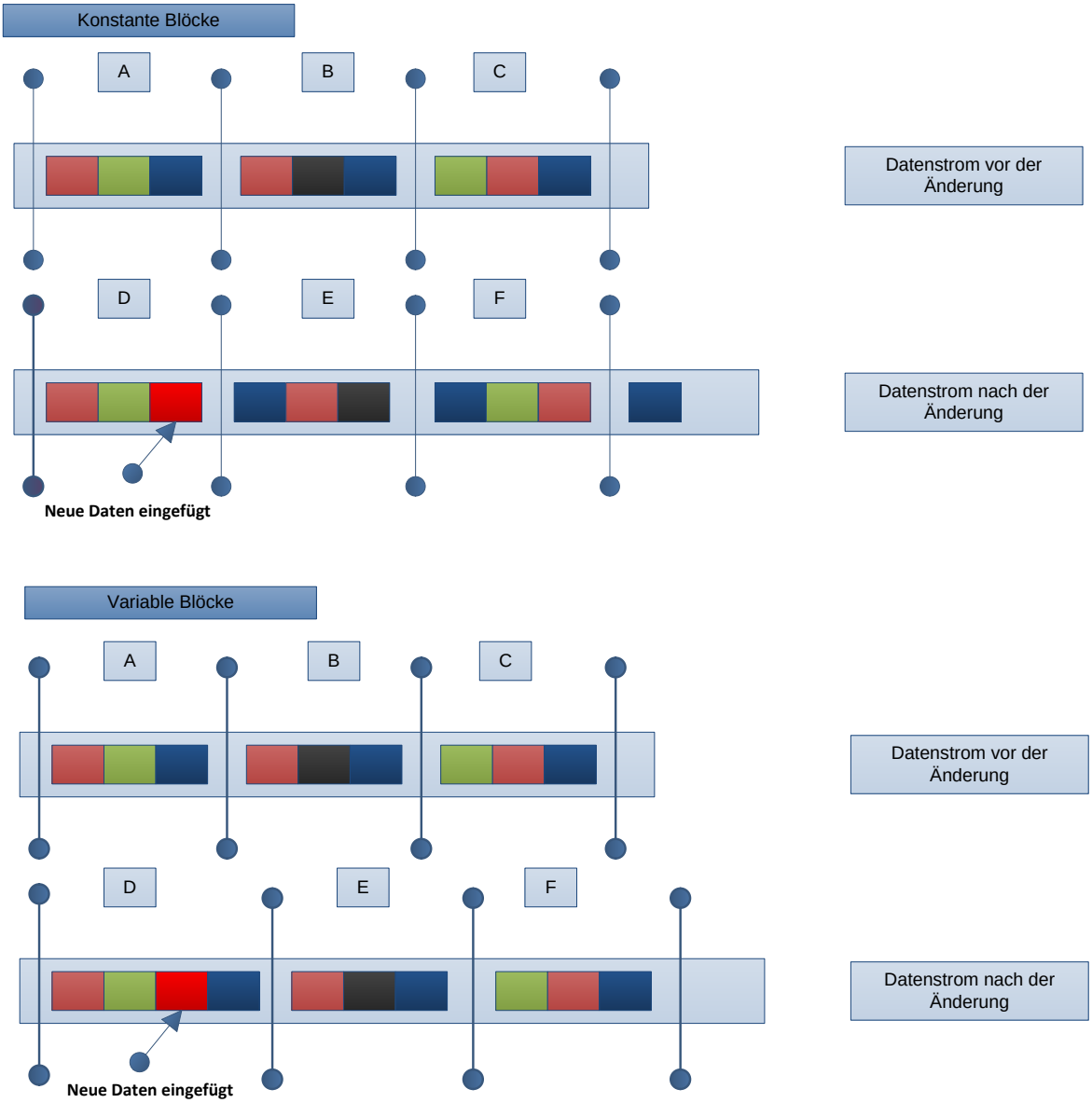


Abbildung 10: Variable Blöcke

Beim System mit der konstanten Blockgröße entstehen vier neue Blöcke, bei der Variante mit der variablen Blockgröße kann der Algorithmus die Größe vom ersten Block verändern, um die darauffolgenden Blöcke in ihrem Originalzustand zu belassen. Dadurch entsteht ein neuer größerer Block und die darauffolgenden bleiben unangetastet.

2.6 DEDUP-Methoden

Viele Produkte lassen dem Anwender die Wahl, wann und wie eine Datendeduplizierung durchgeführt wird. Eine der wichtigsten Entscheidungen, die sich direkt auf die Verarbeitungsgeschwindigkeit auswirkt, ist die Auswahl zwischen Inband- oder Outband-Verarbeitung.

2.6.1 DEDUP-Inband

Wie in der Diplomarbeit [DATA2009] beschrieben, werden bei der Inband-Datendeduplizierung die redundanten Daten vor der Speicherung entfernt. Das DEDUP-System empfängt den Datenstrom, entfernt redundante Daten noch im Hauptspeicher innerhalb des DEDUP-Systems und schreibt die einzigartigen Daten in das Storage-System.

Dadurch, dass die Deduplizierung vor dem Schreiben der Daten stattfindet, muss das DEDUP-System mit genügend Ressourcen ausgestattet sein, um diese bewältigen zu können. Hauptsächlich werden bei diesem Prozess der Hauptspeicher und die CPU gefordert. Sind die Ressourcen begrenzt, wirkt sich dies direkt auf den Datendurchsatz aus. Ein Vorteil ist die sofortige Platzeinsparung, es muss kein zusätzlicher Speicherplatz für den DEDUP-Prozess bereitgestellt werden.

Inband-Datendeduplizierung wird meist in einer Datensicherungsumgebung eingesetzt, in der der benötigte Datendurchsatz leicht ermittelt werden kann.

2.6.2 DEDUP-Outband

Im Gegensatz zum DEDUP-Inband-Verfahren wird beim Outband-Verfahren die Deduplizierung zu einem gewünschten Zeitpunkt durchgeführt. Dieser nachgelagerte Prozess kann dazu genutzt werden, die Datendeduplizierung bei geringer Last am Storage-System durchzuführen. Ist es möglich, ein ausreichend großes Zeitfenster für die Datendeduplizierung zu finden, erhält man die Vorteile der Datenreduktion eines DEDUP-Systems und die volle Leistung des Storage-Systems, wenn der DEDUP-Prozess nicht ausgeführt wird.

Bei dieser Variante erhält man ein Storage-System, das zu bestimmten Zeiten die volle Performance liefert und zusätzlich den Vorteil einer reduzierten Datenmenge.

Wichtig beim Einsatz dieser Art der Duplizierung ist ein ausreichend großer Datenspeicher für die Zwischenlagerung der nicht verarbeiteten Daten. Solange der DEDUP-Prozess nicht gestartet wurde, müssen die deduplizierten Daten und die nicht deduplizierten gespeichert und verwaltet werden. Beim Inband-Verfahren ist es durch die sofortige Verarbeitung nicht notwendig, diesen Speicherplatz zur Verfügung zu stellen.

2.7 Protokolle

2.7.1 Network Filesystem

Um Daten über mehrere Rechnersysteme zu verarbeiten, wurde das Protokoll NFS [RFC 1094] entwickelt. Ein NFS-Server verwaltet ein oder mehrere Dateisysteme, die von den NFS-Clients eingehängt werden. Die eingehängten NFS-Dateisysteme verhalten sich wie lokale Dateisysteme, dadurch ist es möglich, NFS-Dateisysteme für die Speicherung der Konfigurationsdaten einer Maschine zu verwenden. Mit dieser Funktion kann ein NFS-Server die Verwaltung und Konfiguration mehrerer Maschinen zentralisieren. [NFS1991]

2.7.1.1 Anstoß für NFS

In den ersten großen Rechnernetzen wurden einige wenige starke Computer verwendet, mit denen die Benutzer über ein Terminal agierten. Der Datenaustausch zwischen den Großrechnern konnte über ein Kopierprogramm innerhalb der Maschine bewerkstelligt werden.

Heutzutage sind die Umgebungen stark gewachsen, die einigen wenigen Großrechner werden durch viele Serversysteme ersetzt und die Benutzer greifen über eine Workstation auf diese zu. Ein Datenaustausch über einen Kopierbefehl zwischen mehreren Rechnern ist aufwendig und fehleranfällig, da er zu mehreren unterschiedlichen Versionsständen der Daten führen kann. Aus dieser Anforderung heraus, Daten auf eine einfache Weise über mehrere Systeme bereitzustellen, wurde das Protokoll NFS von der Firma SUN entwickelt. [NFS1998]

NFS wird meist in UNIX-basierenden Umgebungen eingesetzt. Für Windows-Umgebungen wird ein ähnliches Protokoll mit dem Namen CIFS[7] verwendet, das auf dem Protokoll SMB[8] aufbaut. [WIKI2012]

2.7.1.2 Aufbau

Bei NFS handelt es sich um ein Protokoll, das in der Anwendungsschicht operiert und auf den Protokollen der unteren Schichten aufsetzt. Abbildung 11 zeigt die Einordnung vom NFS-Protokoll im ISO/OSI-Modell. [RFC1094]

Schichten	NFS-Protokoll
Anwendungsschicht	NFS
Darstellungsschicht	XDR
Sitzungsschicht	RPC
Transportschicht	TCP oder UDP
Vermittlungsschicht	IP
Sicherungsschicht	Ethernet-MAC
Bitübertragungsschicht	Ethernet-Physical

Abbildung 11: ISO/OSI-NFS

In der Darstellungsschicht findet das Protokoll *external data presentation* (XDR) [RFC1014] Anwendung. XDR stellt sicher, dass Daten hardware- und betriebssystemunabhängig übertragen werden. Entwickelt wurde XDR von der Firma SUN und wird haupt-

[7] CIFS Common Internet File System wurde 1996 von Microsoft eingeführt und ist eine Erweiterung vom SMB.

[8] SMB Server Message Block stellt die Basis für CIFS und wird für Kommunikation von Datei-, Druck- und andere Serverdienste in Netzwerken verwendet.

sächlich im Bereich NFS eingesetzt. Die Sitzungsschicht wird mit *remote procedure call* (RPC) abgebildet und wird im nächsten Abschnitt genauer beschrieben.

2.7.1.3 Architektur

NFS setzt auf ein Client/Server-Modell auf, das RPC voraussetzt. RPC ermöglicht laut [NFS1998] Prozesse, die auf entfernten Maschinen ausgeführt werden, lokal erscheinen zu lassen. Dieses Verfahren nutzt entfernte Ressourcen, die auf dem lokalen Rechner nicht zur Verfügung stehen, um diese als lokale Ressource einzubinden. Ein Rechner, der eine Ressource bereitstellt, wird zum Server, der Rechner, der diese Ressource nutzt, wird zum Client.

Als Transportprotokoll kann TCP oder UDP für RPC-Dienste genutzt werden. Abbildung 12 zeigt die Kommunikation eines NFS-Clients mit dem NFS-Server. Wird bei der Übertragung UDP eingesetzt, übernimmt RPC die Sicherung der Übertragung.

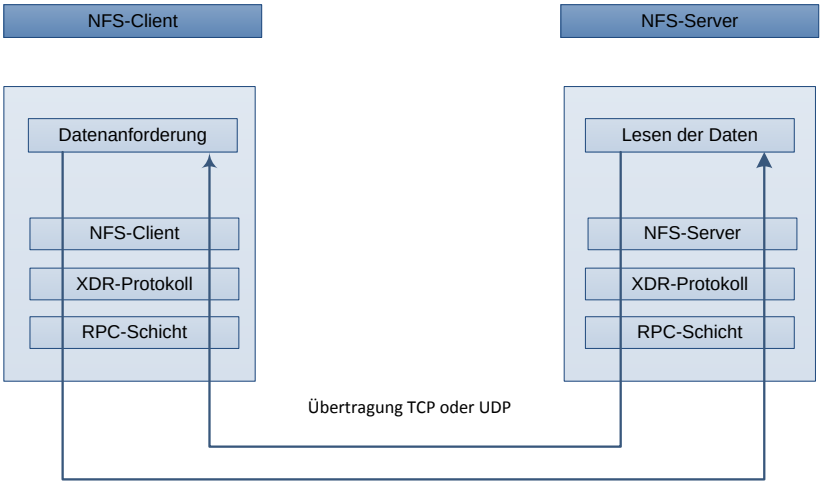


Abbildung 12: NFS-Server/Client

Die Kommunikation zwischen NFS-Server und NFS-Client findet bis zur Version 3 wahlweise über TCP oder UDP statt. Ab der NFS-Version 4 ist UDP nicht mehr möglich und es kommt ausschließlich TCP zum Einsatz.

2.7.2 ISCSI

In einem professionellen Datenspeicherungsumfeld ist das *small computer system interface* (SCSI) nach wie vor das meisteingesetzte Protokoll zur Übertragung von Blockdaten zwischen einem Server und einem Endgerät. SCSI wird in vielen Bereichen der IT verwendet, um beispielsweise einzelne Festplatten, Storage-Systeme und Bandlaufwerke an Server-Systeme anzubinden. Über mehrere Generationen hinweg hat sich SCSI von einer parallelen Schnittstelle zu der aktuellen seriellen Serial-Attached-SCSI, kurz SAS genannt, weiterentwickelt.

Diese Schnittstelle bietet derzeit eine Transferrate von 6 Gbit/s und kann durch Nutzung von zwei Kanälen in Verbindung mit einem Festplattensystem auf eine Transferrate von

12 Gbit/s erhöht werden. Für Bandlaufwerke steht die Nutzung von zwei Kanälen nicht zur Verfügung. [WS2012]

Wie die vorangegangenen SCSI-Generationen ist die SAS-Schnittstelle nur für kurze Strecken geeignet. Eine Kabellänge von max. 25 Metern wird mit der aktuellen Technologie erreicht. Aus diesem Grund findet die SAS-Schnittstelle meist innerhalb von Speichersystemen und Serversystemen im Backbone-Bereich Anwendung. Wird eine Anbindung vom Server an weiter entfernte Endgeräte gefordert, wird das Fibre-Channel-Protokoll (FC-Protokoll[9]) für hohe Geschwindigkeitsansprüche oder als Alternative NFS/CIFS eingesetzt.

2.7.2.1 Anstoß für ISCSI

Das FC-Protokoll setzt eine eigene Infrastruktur aus Hostbusadapter und je nach Ausprägung FC-Switches voraus. Dadurch, dass eine eigene Infrastruktur aufgebaut werden muss, ist auch der Betrieb dieser Infrastruktur mit zusätzlichen personellen Aufwänden verbunden.

Aus diesen Gründen wurde als Konkurrenz zum FC-Protokoll das ISCSI-Protokoll ins Leben gerufen. Diese Technologie verbindet das SCSI-Protokoll mit der Internet-Protokollfamilie. Durch diese Kombination wird es möglich, SCSI-Geräte über die meist vorhandenen Ethernet-basierenden TCP/IP-Netzwerke zu betreiben. Der Einsatz von ISCSI kann die personellen Kosten gering halten, da die meist vorhandenen Ressourcen genutzt oder nur erweitert werden müssen. Ein Aufbau neuer Ressourcen und Betreuung einer neuen fremdartigen Infrastruktur entfällt. [ISCSI2006]

Durch die Einführung neuer Netzwerkkomponenten, die eine Übertragungsrate bis zu 10Gbit/s und zukünftig 40Gbit/s unterstützt, kann ISCSI im Bereich der Geschwindigkeit und Zugriffszeit zum FC-Protokoll aufschließen.

2.7.2.2 Aufbau

ISCSI wird im ISO/OSI-Referenzmodell in der Verbindungsschicht abgebildet. Die höhere Schicht ermöglicht es, die Routingfunktionen und Paketwiederherstellung vom TCP/IP-Protokoll zu nutzen. Durch den Einsatz von TCP/IP können ISCSI-Verbindungen innerhalb eines LAN wie auch in einem MAN oder WAN verwendet werden.

Die Abbildung 13 zeigt die Einordung des ISCSI-Protokolls im ISO/OSI-Referenzmodell. SCSI-Befehle, sogenannte CDBs[10], werden in der Darstellungsschicht generiert und vom ISCSI-Protokoll in der Sitzungsschicht entgegengenommen. Die CDBs werden danach in TCP/IP-Pakete gekapselt und schlussendlich über das Ethernet-Datennetz übertragen.

---

[9] Fibre Channel ist ein Datenübertragungsprotokoll, das für große Datenmengen und hohe Geschwindigkeiten konzipiert wurde.

[10] CDB Command Description Block ist die kleinste Einheit, die im SCSI-Protokoll verwendet wird.

Schichten	ISCSI-Protokoll
Anwendungsschicht	
Darstellungsschicht	SCSI
Sitzungsschicht	ISCSI
Transportschicht	TCP oder UDP
Vermittlungsschicht	IP
Sicherungsschicht	Ethernet-MAC
Bitübertragungsschicht	Ethernet-Physical

Abbildung 13: ISO/OSI-ISCSI

2.7.2.3 Architektur

In einer ISCSI-Umgebung wird ein Gerät als ISCSI-Ziel oder ISCSI-Target bezeichnet, das eine ISCSI-LUN[11] bereitstellt. Das Hostsystem, das ein ISCSI-Target verwendet, wird als ISCS-Initiator bezeichnet. In der Abbildung 14 wird ein vereinfachter Aufbau einer IS-CSI-Umgebung dargestellt.

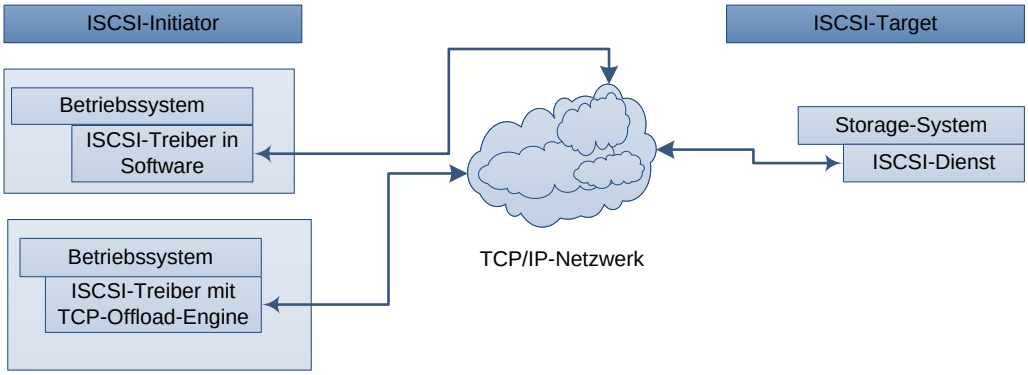


Abbildung 14: ISCSI-Aufbau

Um ein ISCSI-Gerät verwenden zu können, muss der Client in einem ISCSI-Netzwerk über einen ISCSI-Initiator verfügen. Dieser ISCSI-Initiator entspricht dem Hostbusadapter in einer SCSI-Umgebung. Der ISCSI-Initiator kann als Hardware mittels einer PCI-X oder PCIe-Karte oder als Software-Adapter implementiert sein.

2.7.2.4 Verbindungsaufbau

Die ISCSI-Session ist die höchste Schicht im ISCSI-Protokoll und wird als logische Ver-knüpfung zwischen einem ISCSI-Initiator und dem ISCSI-Target verwendet. Innerhalb dieser Verbindung werden Daten sowie Befehle übertragen. Man unterscheidet zwischen zwei Arten von ISCSI-Sessions, *normal operation session* und *discovery session*.

[11] LUN – Logical Unit Number, ein ISCSI-Target kann eines oder mehrere LUNs bereitstellen. Über die LUN kann jedes dieser Geräte einzeln angesprochen werden.

Um ISCSI-Geräte zu erkennen und einzubinden, wird die *discovery session* verwendet, die darauffolgenden Operationen wie die Daten- und Befehlsübertragung wird von der *normal operation session* durchgeführt.

Jede ISCSI-Session wird über eine eigene Session-ID der SSID identifiziert. Die SSID besteht immer aus zwei Teilen, dem *initiator-side identifier* (ISID) für den ISCSI-Initiator und dem *target-side identifier* (TSID) für das ISCSI-Target.

Das TCP/IP-Protokoll garantiert die Übertragung der Daten in der richtigen Reihenfolge, die vom SCSI-Protokoll und damit auch vom ISCSI-Protokoll gefordert wird. Um die Geschwindigkeit und die Zugriffszeit zu erhöhen, wurde das ISCSI-Protokoll für die Verwendung mehrerer TCP/IP-Verbindungen zur selben Zeit erweitert. Durch diese Erweiterung ist die Möglichkeit, die Pakete in der richtigen Reihenfolge abzuliefern, mithilfe des TCP/IP-Protokolls nicht mehr gegeben, da TCP/IP die Reihenfolge von Paketen nur pro Verbindung kontrollieren kann. Um dieses Problem zu umgehen, wurde die *protocol data unit* (PDU) um ein Feld erweitert, das für die Reihenfolge der Pakete genutzt wird.

2.7.2.5 ISCSI-Protocol Data Unit

Die kleinste Übertragungseinheit des ISCSI-Protokolls ist die PDU. PDUs werden für die Kommunikation zwischen dem ISCSI-Initiator und dem ISCSI-Target eingesetzt. Jede PDU besteht aus mehreren Header gefolgt von Nullen oder Daten.

Das *basic header segment* (BHS) ist das erste Segment in einer PDU. Dieses Segment ist notwendig für eine funktionsfähige PDU, alle weiteren Segmente sind optional. Das BHS besteht aus einer fixen Länge von 48 Bytes und beinhaltet Informationen der CDB, ISCSI-LUN usw.

Für die verschiedenen Operationen in der ISCSI-Kommunikation sind unterschiedliche PDU-Typen verantwortlich, die sich durch den *operational code* (Op-Code) im BHS, das ein Teil der PDU ist, unterscheiden. In der nachfolgenden Abbildung 15 wird die PDU in einem TCP/IP-Segment dargestellt.

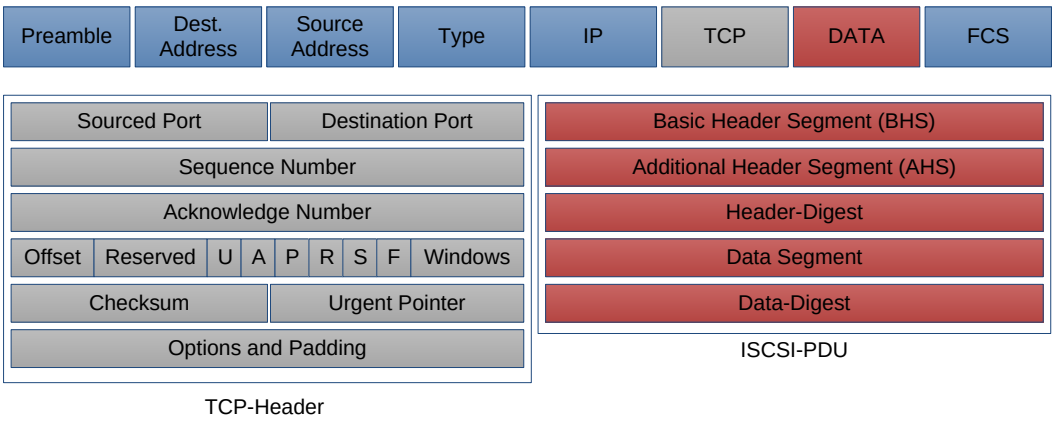


Abbildung 15: ISCSI OP-Code

2.8 Hashfunktionen

In vielen Bereichen der IT werden Hashfunktionen oder Streuwertfunktionen eingesetzt. Sie dienen allgemein dazu, aus einer beliebig langen Nachricht einen kleineren Hashwert zu erzeugen. Siehe Abbildung 16.

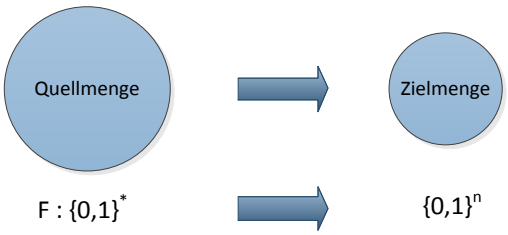


Abbildung 16: Hashfunktion

Aus einer theoretischen beliebig langen Eingabe wird eine Ausgabe mit einer konstanten Länge definiert. Eingeschränkt wird die Größe der Quellmenge durch die eingesetzte Hashfunktion. [WH2012]

2.8.1 Funktionsweise einer Hashfunktion

Eine sehr einfache Art einer Hashfunktion stellt die Quersumme da. Wenn von der Zahl n = 345682 die dezimale Quersumme q(n) berechnet wird, müssen die Ziffernwerte addiert werden. [WQ2012]

q(n) = 3+4+5+6+8+2 = 28

Aus der Quellmenge 346682 wird die Zielmenge 28 berechnet. Diese Hashfunktion bietet keinen Schutz gegen den Zifferntausch oder Zahlensturz, da z. B. auch die Zahl 436682 der Quersumme 28 entspricht. Wenn aus verschiedenen Quellmengen die Hashfunktion gleiche Hashwerte generiert, handelt es sich um eine Hashkollision. Im Abschnitt 2.8.3 wird darauf näher eingegangen.

Ein weiteres Beispiel aus der Praxis bietet die ISBN[12]. Bei der ISBN handelt es sich um eine eindeutige Kennzeichnung für Bücher und sie wird durch eine Prüfsumme geschützt. Diese Prüfsumme wird über eine Hashfunktion erstellt. [WI2012]

Einer der häufig eingesetzten Bereiche für Hashfunktionen ist die Passwortverschlüsselung. Anstatt die Passwörter im Klartext abzulegen, wird die Ausgabe der Hashfunktion gespeichert. Um die Richtigkeit des Passworts zu überprüfen, wird die Eingabe des Benutzers mit der Hashfunktion verarbeitet und der aktuelle Wert mit dem gespeicherten Wert verglichen. Handelt es sich um den gleichen Wert, wird die Eingabe als korrekt erkannt. Für diese Art der Anwendung eignen sich laut [KIT2008] Einweg-Hashfunktionen, bei denen es praktisch nicht möglich ist, vom Hashwert auf den Eingabewert rückzuschließen, und dadurch die Passwörter geschützt sind.

[12] ISBN International Standardbuchnummer: Eindeutige Kennzeichnungsnummer für Bücher, die im Buchhandel eingesetzt wird.

Weitere Einsatzgebiete für die Hashfunktion sind die Kontrolle bei der Datenübertragung und Speicherung auf ihre Integrität mittels CRC[13] oder die Kontrolle bei digitalen Signaturen.

2.8.2 Hashfunktionen in der Daten-Deduplizierung

Die Hashfunktion wird beim DEDUP-System eingesetzt, um die verarbeiteten Daten eindeutig zu kennzeichnen. Je nach Granularität werden für die Berechnung des Hashwerts ganze Dateien oder Teile davon verwendet.

Bei einem DEDUP-System, das auf Blockebene arbeitet, wird von jedem Block ein Hashwert oder Fingerprint erzeugt. Der Vergleich der Fingerprints entscheidet, ob es sich bei dem Block um ein Unikat oder Duplikat handelt.

Der Fingerprint besitzt je nach eingesetztem Algorithmus nur einen Bruchteil der Größe vom dem verarbeiteten Block. Die Hashfunktion Tiger[14] erstellt aus einer Eingangsmenge einen 192 Bit großen Hashwert. Wird eine Blockgröße von 4 KB angenommen, ist der Hashwert um den Faktor 170 kleiner.

Die Reduzierung der Daten bietet zwei Vorteile, der Vergleich der Blöcke ist bei einem 192 Bit großen Hashwert schneller als bei einem 4 KB großen Block und die Verwaltung der Blöcke benötigt wenig Speicherplatz.

2.8.3 Hashkollisionen

Eine Hashkollision tritt auf, wenn zwei unterschiedliche Eingangswerte den gleichen Hashausgangswert erzeugen. Die Quersumme, wie in Abschnitt 2.8.3 beschrieben, ist ein Beispiel für einen Hashalgorithmus, bei dem sehr einfach Kollisionen auftreten können.

Kollisionsresistenz ist ein Qualitätsmerkmal einer Hashfunktion, das Finden oder Auftreten einer Hashkollision sollte praktisch nicht durchführbar sein. Eine hundertprozentige Vermeidung von Kollisionen ist nicht möglich, es kann nur die Wahrscheinlichkeit mit unterschiedlichen Methoden beeinflusst werden. Eine einfache Methode ist die Erhöhung des Ausgangswerts der Hashfunktion. Die Wahrscheinlichkeit einer Kollision ist bei einem Algorithmus, der 192 Bit für den Ausgangswert verwendet, geringer als bei einem, der 128 Bits verwendet.

Im Bereich DEDUP ist eine Hashkollision gleichbedeutend mit Datenverlust, da ein Block, der ein Unikat ist, als Duplikat erkannt wird. Hersteller gehen verschiedene Wege, um dieses Problem zu lösen. Die sicherste Methode, Datenverluste durch Hashkollision auszuschließen, ist der Bitvergleich bei jedem gefundenen Duplikat. [COL2010]

---

[13] CRC *cyclic redundancy check* wird eingesetzt, um Prüfsummen zu erzeugen, die zur Fehlererkennung einer Datenübertragung oder Datenspeicherung dienen.

[14] Tiger ist eine Hashfunktion, die beim Open-Source-Projekt SDFS angewendet wird.

3 Softwareauswahl

Dieses Kapitel befasst sich mit der Definition der Anforderungen an eine DEDUP-Software und der Softwareauswahl. Die Anforderungskriterien werden in Hinblick auf die Unterstützung einer VMware-ESX-Umgebung herausgearbeitet.

3.1 Detailbeschreibung

Enterprise-Umgebungen stellen hohe Anforderungen an Stabilität, Funktionalität, Geschwindigkeit und Administrierbarkeit für jegliche Komponenten in einer IT-Umgebung. Für einen Outsourcing-Betreiber sind die wichtigsten Ziele, IT-Dienstleistungen für den Kunden schnell und stabil bereitzustellen und sie zusätzlich kostengünstig und einfach in der Wartung zu gestalten.

Um eine DEDUP-Software als Datenspeicher in einer VMware-ESX-Umgebung erfolgreich einsetzen zu können, müssen zwei grundlegende Funktionen erfüllt sein. Erstens müssen die geforderten Protokolle der ESX-Umgebung unterstützt werden und zweitens muss die DEDUP-Funktion transparent arbeiten.

Datenspeicher, auch Storage-Systeme genannt, können lokal oder entfernt an einen ESX-Server angebunden werden. Bei der lokalen Anbindung handelt es sich meist um die lokale Festplatte im ESX-Server. Für produktive Umgebungen ist dieser Datenspeicher nicht geeignet, da wichtige Funktionen wie die Migration von einer virtuellen Maschine während des Betriebs auf einen anderen Datenspeicher nicht möglich sind. Mit dieser Einschränkung ist ein hochverfügbarer Betrieb einer virtuellen Maschine nicht möglich.

Ein entfernter Datenspeicher kann über die Protokolle ISCSI oder NFS in einem TCP/IP-Netzwerk oder über das FC-Protokoll in einem FC-Netzwerk angebunden werden. Mit diesen Storage-Systemen ist es möglich, virtuelle Maschinen hochverfügbar zu konfigurieren. Die Option der Migration einer virtuellen Maschine im Betrieb ist gegeben.

Wenn diese grundlegenden Anforderungen erfüllt sind, kann eine DEDUP-Software nach den definierten Anforderungskriterien in Abschnitt 3.2 überprüft werden.

3.2 Anforderungskriterien

Die nachfolgende Tabelle stellt eine Übersicht über die definierten Anforderungskriterien dar. Jedes Anforderungskriterium ist mit Fragen hinterlegt, die in Kapitel 4 in einer Praxisumgebung überprüft und bewertet werden. Die Anforderungskriterien sind nach ihrer Relevanz absteigend geordnet.

Anforderungskriterien	Beschreibung
Stabilität	Wie verhält sich die Software im Betrieb? Können Datenverluste auftreten?
Performance	Wie hoch sind die Geschwindigkeitsverluste einer ESX-Umgebung mit einem Storage-System, das DEDUP einsetzt, im Vergleich zu einer ESX-Umgebung ohne DEDUP?
Datenreduktion	Wie viel Speicherplatz benötigt eine und mehrere virtuelle Maschinen in einer ESX-Umgebung mit Datendeduplizierung oder ohne Datendeduplizierung?
Konfiguration	Wie hoch ist der Aufwand für die Konfiguration der DEDUP-Software? Können Konfigurationen während des Betriebs geändert werden?
Installationsroutine	Welche Voraussetzungen müssen geschaffen werden, um ein DEDUP-System einzusetzen? Wie viel Aufwand liegt in der Installation?
Dokumentation	Ist das System dokumentiert? Existieren Handbücher und Installationsanleitungen?
Support	Welche Unterstützung wird geboten?

Tabelle 1: Anforderungskriterien

Nach der Überprüfung der Anforderungskriterien in der Laborumgebung, wird eine Bewertung der Software vorgenommen. Diese Bewertung wird auf Basis der Ergebnisse in der Laborumgebung erstellt.

3.3 LessFS

LessFS wird von Mark Ruijter entwickelt. Er startete das Projekt 2009. Veröffentlicht wurde das Programm unter der GPL[15]-Lizenz in der Version 3. Die Programmiersprache C wird für das Projekt eingesetzt. Der Quelltext ist beim Anbieter SourceForge hinterlegt.

Projektname	LessFS
Entwickler	Mark Ruijter
Projektstart	April 2009
Lizenz	GNU GPL v3
Programmiersprache	C
Homepage	http://www.lessfs.com
Quelltext	http://sourceforge.net/projects/lessfs/files/

Tabelle 2: Projekt LessFS

Die Entwicklung von LessFS konzentriert sich hauptsächlich auf die DEDUP-Funktionalität. Optimierungen der Geschwindigkeit und die Deduplizierungsleistung stehen im Vordergrund. Technologien wie Tier[16] und EPRD[17] unterstreichen diese Art der Projektentwicklung.

Der Einstieg und Einsatz von LessFS wird durch die sporadische Webseite erschwert. Eine Vielzahl von Begriffen und Konfigurationsschritten wird auf dieser Webseite präsentiert. Dokumentationen oder eine allgemeine Einleitung fehlen. Weiterführende Links setzen eine funktionierende LessFS-Installation voraus, ohne näher auf den Weg dorthin einzugehen.

Beispielhafte LessFS-Implementierungen werden auf einschlägigen Webseiten vorgestellt, die aber keine direkte Verbindung zum Projekt besitzen. Installation und Konfiguration sind nur über tiefer gehende Linux-Kenntnisse möglich, da Mark Ruijter ausschließlich den Quellcode und keine Binärpakete zur Verfügung stellt. Die Anwender oder Linux-Distributoren sind gefordert, geeignete Binärpakete und Installationsanleitungen zu erstellen.

Obwohl der Einsatz von LessFS durch diverse Hürden erschwert wird, ist das Projekt durchwegs positiv auf den einschlägigen Internetseiten und in Magazinen präsent (siehe

[15] GPL General Public License ist eine Softwarelizenz, die von der Free Software Foundation (FSF) veröffentlicht wurde.

[16] Bei Tier handelt es sich um ein Linux Kernel Module, das unterschiedliche Datenspeichertypen zu einem zusammenfasst (siehe http://sourceforge.net/projects/tier/).

[17] Bei EPRD handelt es sich um ein Speichergerät, das im Hauptspeicher eines Linux-Systems abgebildet wird (siehe http://sourceforge.net/projects/eprd/). Kann mit dem Projekt Tier kombiniert werden.

http://www.linux-community.de/Internal/Artikel/Print-Artikel/LinuxUser/2010/11/Das-deduplizierende-Dateisystem-LessFS).

3.4 SDFS

Mark Silverberg entwickelt seit April 2010 das Programm SDFS. SDFS besteht aus mehreren Komponenten, die zusammen ein Deduplizierungssystem mit einem Webinterface für die Überwachung und Konfiguration bilden.

Das Projekt SDFS unterteilt sich in das Dateisystem SDFS, das für die Deduplizierung und Präsentation der deduplizierten Daten verantwortlich ist, und die Software SDFS-Management-Konsole, die eine grafische Schnittstelle über ein Webinterface bietet.

Projektname	SDFS
Entwickler	Mark Silverberg
Projektstart	April 2010
Lizenz	GNU GPL v2
Programmiersprache	Java
Homepage	http://opendedup.org/
Quelltext	http://code.google.com/p/opendedup/downloads/list

Tabelle 3: Projekt SDFS

Präsentation und Einstieg in das Projekt SDFS unterscheiden sich immens gegenüber dem Projekt LessFS. Die Seite wird von einem zentral eingebetteten Video dominiert, das die Funktionalität von SDFS in der Kombination mit einem ESX-Server demonstriert. Darunter finden sich wichtige Eckpunkte und Leistungsdaten der Software.

Die linke Seite wird für die Navigation verwendet. Dort finden Interessierte Dokumentationen zu den einzelnen unterstützten Plattformen und zusätzliche „*quick start guides*“, um SDFS innerhalb einer kurzen Zeit in Betrieb zu nehmen. Die rechte Seite wird für Neuigkeiten, die SDFS betreffen, genutzt. Wie zum Beispiel die Ankündigung einer neuen Version der SDFS-Software.

Im Bereich „Download“ stellt Mark Silverberg neben dem obligatorischen Quellcode zusätzlich Binärpakete für die größten Linux-Distributoren und das Betriebssystem Windows bereit. Anwender, die eine ESX-Umgebung betreiben, können eine vorkonfigurierte SDFS-Appliance in ihre ESX-Umgebung herunterladen und einbinden. Das Angebotene Format dieser virtuellen Maschinen ist mit einer ESX-Umgebung kompatibel.

SDFS bietet auf der Einstiegsseite eine Fülle von Informationen, die auf den ersten Blick den Interessenten abschrecken. Durch die klare Strukturierung der Projektseite und guten Dokumentationen wird der Ersteindruck entschärft.

3.5 Softwareauswahl

Für die Auswahl des Projekts wird eine Tabelle verwendet, die es ermöglicht, die definierten Auswahlkriterien zu bewerten und zu gewichten. Ein Punktesystem, das eine Bewertung von 1 bis 3 Punkten pro Kriterium vorsieht, wird eingesetzt. Eine Gewichtung der Kriterien wird zusätzlich vorgenommen, um diese nach ihrer Relevanz zu priorisieren. Für die Wertigkeit sind zusätzlich Punkte zwischen 1 und 3 vorgesehen. Die Gesamtpunktezahl eines Kriteriums errechnet sich durch die Multiplikation der Bewertung mit der Gewichtung.

Auswahlkriterien, die für eine Integration in eine ESX-Umgebung wichtig sind, erhalten eine höhere Gewichtung als die restlichen Kriterien.

Auswahlkriterium	Gewichtung	SDFS	LessFS
ESX-Integration	3	3	2
Dokumentation	2	3	1
Usability	2	2	1
DEDUP-Inband	2	1	1
DEDUP-Outband	2	3	0
Installationsprozess	1	2	1
Konfiguration	1	1	1
Bewertung	Max. Punkte 36	30	14

Tabelle 4: Projekt-Bewertung

Beide Softwareprojekte bieten ein Dateisystem mit einer DEDUP-Funktion an. Funktional sind sie bis auf ein paar wenige Punkte gleichwertig.

Für den Betrieb von LessFS wird ein Linux-Betriebssystem vorausgesetzt. SDFS unterstützt durch die Programmiersprache Java Linux-Betriebssysteme wie auch Windows-Betriebssysteme. In der Grundinstallation ist die Datendeduplizierung auf das lokale Betriebssystem, auf dem sie konfiguriert und installiert wurden, beschränkt.

Keines der zwei Projekte bietet eine integrierte Möglichkeit, das Dateisystem über NFS oder ISCSI für andere Systeme zur Verfügung zu stellen. Der Anwender ist für diese

Funktionalität verantwortlich. Über die Bordmittel, die ein Windows-Betriebssystem bietet, kann eine SDFS-Installation über das Protokoll CIFS weitergegeben werden.

Innerhalb eines Linux-Betriebssystems sind je nach eingesetzter Linux-Distribution NFS, CIFS oder ISCSI möglich.

LessFS bietet im Gegensatz zu SDFS weitere Softwareprojekte an, um die Performance der Datendeduplizierung zu erhöhen. Dabei handelt es sich um weiterführende Projekte, die ein Linux-Betriebssystem mit zusätzlichen Möglichkeiten ausrüsten, um LessFS zu erweitern und zu beschleunigen.

Diese zusätzlichen LessFS-Projekte werden wie LessFS selbst in Form von Quelltexten angeboten. Kenntnisse in der Kompilierung von Quellcodes und Administration des Linux-Betriebssystems werden dadurch vorausgesetzt. Der Einstieg ist mit einem hohen Aufwand verknüpft.

Das Projekt SDFS bietet zu der manuellen Installation über Quellcodes und Binärpakete einen weiteren Weg. Mark Silverberg integrierte SDFS in eine Linux-Distribution und entwickelte eine WEB-basierende Verwaltungssoftware, um die grundlegenden Konfigurationen des Linux-Betriebssystems und SDFS zu ermöglichen. Diese Verwaltungssoftware wird unter dem Namen SDFS-Management-Konsole angeboten.

Die Kombination SDFS, Linux-Betriebssystem und Verwaltungssoftware wird auf der Projekthomepage als „SDFS Software Appliance“ angeboten. Diese Appliance kann über wenige Schritte in einer ESX-Umgebung eingebunden werden.

Von außen verhält sich die SDFS-Appliance wie ein Storage-System, das über die Protokolle NFS und ISCSI die SDFS-Volumes zur Verfügung stellt. Die DEDUP-Funktion arbeitet im Hintergrund völlig transparent. Zusätzlich zur Integration von NFS und ISCSI kann die SDFS-Management-Konsole über das Applikations-Interface (API) von ESX auf eine ESX-Umgebung zugreifen. Diese Verbindung wird dazu genutzt, die SDFS-Dateisysteme nach der Erstellung in die ESX-Umgebung einzubinden. Nach dieser Integration kann der Datenspeicher für virtuelle Maschinen genutzt werden.

Die gute Dokumentation, das Angebot der SDFS-Software-Appliance und die enge Verzahnung mit ESX sind die Hauptgründe für die Entscheidung zu Gunsten SDFS.

4 SDFS im praktischen Einsatz

Dieses Kapitel beschäftigt sich mit der Planung und dem Aufbau einer Virtualisierungsumgebung, die SDFS in der vorgefertigten Software-Appliance einsetzt. Die Anforderungskriterien aus Kapitel 3 werden in dieser Umgebung überprüft. Die Kriterien Performance und Datenreduktion werden jeweils anhand dreier Szenarien überprüft.

4.1 Internet Aufbau SDFS-Appliance

SDFS ermöglicht es, Datendeduplizierung transparent in einer ESX-Umgebung einzusetzen. Um dies zu erreichen, wird ein Dateisystem innerhalb der SDFS-Appliance erstellt und über die Protokolle NFS oder ISCSI bereitgestellt. Diese Konfigurationsschritte können über ein grafisches Webinterface der SDFS-Management-Konsole oder auch direkt in der Kommandozeile vorgenommen werden. Die SDFS-Appliance verhält sich von außen betrachtet wie ein Storage-System vom Typ NAS[18].

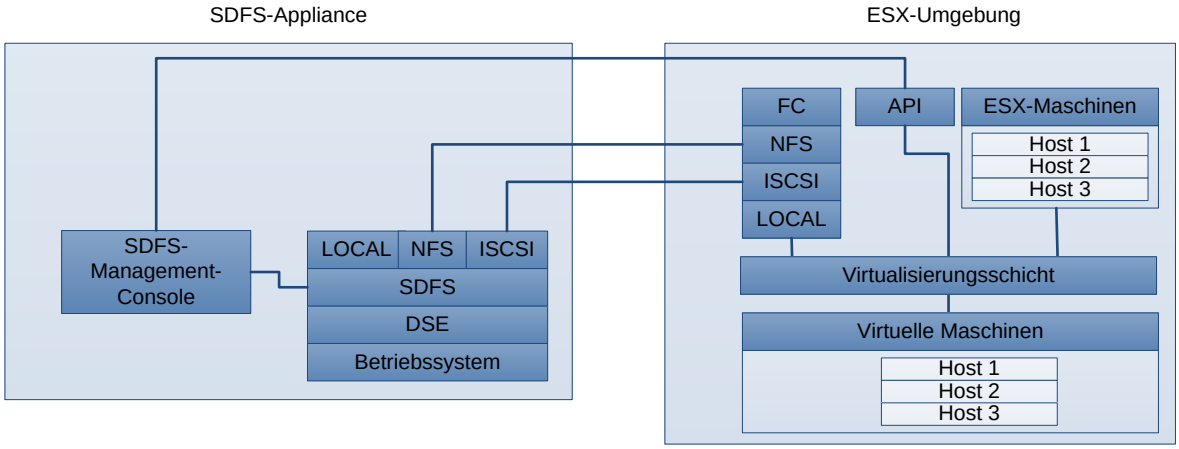


Abbildung 17: Integration SDFS

Abbildung 17 zeigt die Integration der SDFS-Appliance in einer ESX-Umgebung. Über die ESX-API werden die SDFS-Volumes in der ESX-Umgebung registriert und über die Protokolle NFS oder ISCSI angebunden. Das SDFS-Volume wird in der ESX-Umgebung wie ein Standard-*datastore*[19] behandelt. Die DEDUP-Funktion kann von der ESX-Seite nicht festgestellt werden. Die SDFS-Appliance verhält sich völlig transparent zu der ESX-Umgebung.

---

[18] Network Attached Storage (NAS), handelt es sich um ein Storage-System, das über ein TCP/IP-Netzwerk angebunden wird.

[19] *datastore* bezeichnet einen Datenspeicher für virtuelle Maschinen in einer ESX-Umgebung. Dieser kann lokal über SCSI oder meist über NFS, ISCSI oder FC angebunden sein.

4.1.1 Interner Aufbau SDFS

SDFS setzt auf die vorhandenen Funktionen eines Linux- oder Windows-Betriebssystems auf und integriert die DEDUP-Funktion als zusätzlichen Dienst. Diese Art der Integration ermöglicht eine flexible Anbindung an verschiedenste Systeme. Die Nutzung der Software-Appliance ist nicht zwingend erforderlich, SDFS kann, wenn die Anforderungen erfüllt werden, auch in bestehende Betriebssysteme integriert werden.

Durch den Einsatz von Java als Kerntechnologie ist die Software plattformunabhängig und lässt sich in der aktuellen Form auf einem Windows- oder Linux-Betriebssystem betreiben.

Der Aufbau ist modular und wird in nachfolgender Abbildung 18 dargestellt.

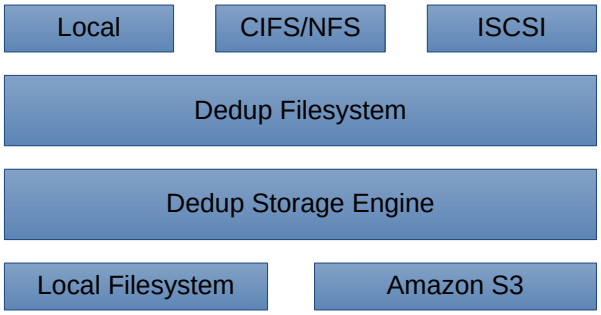


Abbildung 18: Modularer SDFS-Aufbau

Laut [OPEN2012] kann SDFS die DEDUP-Funktion lokal auf der Maschine oder über NFS, ISCSI oder CIFS bereitstellen. Bei dem Modul Amazon-S3 handelt es sich um einen Datenspeicher, der von der Firma Amazon bereitgestellt wird. SDFS bietet die Möglichkeit, die Verwaltung und die Unikate nicht nur auf einem lokalen Dateisystem, sondern auch in einem Amazon-S3-Speicher abzulegen.

4.1.2 SDFS-Module

Die Informationen zu den einzelnen Modulen und ihren Schnittstellen werden den Literaturquellen [OPEN2012] und [OPEN2011] entnommen.

4.1.2.1 SDFS-Volume

Das SDFS-Volume ist der primäre Weg, mit dem Betriebssystem zu kommunizieren. Es dient dazu, die gespeicherten Daten dem Betriebssystem zur Verfügung zu stellen, und verhält sich wie ein Standard-Dateisystem. Ein SDFS-Volume lässt sich über die Protokolle NFS, ISCSI oder CIFS für andere Server oder Clients bereitstellen.

4.1.2.2 SDFS-Filesystem-Service

Eine POSIX[20]-kompatible Ansicht der Daten und Verzeichnisse stellt der SDFS-Filesystem-Service zur Verfügung. Eine zusätzliche Funktion ist die Verwaltung von Metadaten der gespeicherten Daten und Verzeichnisse. Diese beinhalten die Dateigröße und den Ort einer Datei im Verzeichnisbaum. Jedes SDFS-Volume wird von einem eigenständigen SDFS-Filesystem-Service verwaltet.

4.1.2.3 Deduplication Storage Engine

Die Datendeduplizierung wird durch *deduplication storage engine* (DSE) ausgeführt. DSE ist für das Lesen, Speichern und Löschen der Daten verantwortlich. Je nach Konfiguration wird DSE zentral oder pro SDFS-Volume eingesetzt. Bei der zentralen Konfiguration sind die SDFS-Volumes über TCP/IP an einer zentralen Instanz angebunden. Wird ein SDFS-Volume mit den vorgegebenen Parametern erstellt, wird die lokale DSE-Instanz verwendet.

4.1.2.4 Data-Chunks

SDFS unterteilt die Daten in Blöcke, sogenannte Chunks. Aus jedem Chunk wird ein Hashwert/Fingerprint generiert und in einer Tabelle gespeichert. Der Chunk ist die kleinste Einheit, die SDFS verarbeitet, und kann pro SDFS-Volume in einer Größe von 4 KB bis 128 KB definiert sein. Eine nachträgliche Änderung der Chunk-Größe ist nicht möglich. Für die Hashwert-Berechnung wird der Hashalgorithmus Tiger eingesetzt. Dieser kann durch andere Hashverfahren ausgetauscht werden.

4.1.2.5 FUSE

Das Projekt *filesystem in userspace* (FUSE) bildet die Basis vom Dateisystem SDFS und ist für die Präsentation der deduplizierten Daten im Originalzustand verantwortlich. Es handelt sich um ein Softwareprojekt, das es ermöglicht, Dateisysteme im *userspace* zu entwickeln und zu betreiben.

In einem Linux-Betriebssystem werden die Dateisysteme im *kernel space* abgebildet, der nur über definierte Schnittstellen erreichbar ist. Das FUSE-Kernelmodul erlaubt es, ein Dateisystem im *userspace* zu betreiben. Durch die Verschiebung des Dateisystems vom *kernel space* in den *userspace* muss auf spezifische Eigenschaften von Hardware und Kernel keine Rücksicht genommen werden. [TAN2009]

FUSE-Dateisysteme lassen sich mit den Bordmitteln, die ein Betriebssystem zur Verfügung stellt, nicht einhängen. Spezielle Befehle, die in den FUSE-Tools enthalten sind, werden benötigt, um den Zugriff auf das Dateisystem zu ermöglichen. Das eingehängte FUSE-Dateisystem kann über ISCSI, CIFS oder NFS weitergegeben werden.

[20] POSIX Portable Operating System Interface ist eine internationale Norm für ein standardisiertes Application Programming Interface.

Die Abbildung 19 zeigt die Integration von SDFS, FUSE und dem Linux-Betriebssystem.

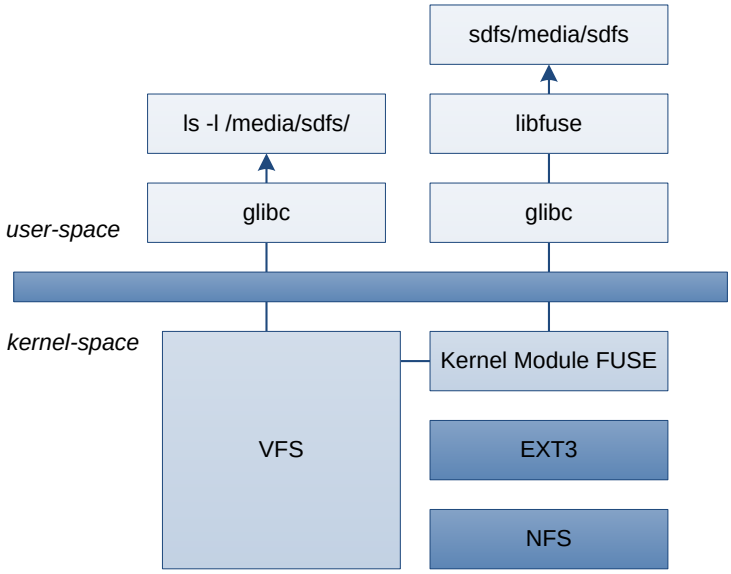


Abbildung 19: FUSE mit SDFS

EXT3 ist ein Linux-Standarddateisystem, das im *kernel space* abgebildet wird. SDFS setzt über das Kernel Module FUSE auf das Dateisystem EXT3 auf. Dadurch kann SDFS die DEDUP-Funktionalität anbieten, ohne auf die Hardware oder Kernel-Eigenschaften Rücksicht zu nehmen.

4.2 Überblick Evaluierungsaufbau

Für die Evaluierung der Deduplizierungssoftware SDFS wurde eine Laborumgebung aufgebaut, die bis auf die Dienste DNS[21] und NTP[22] getrennt von der produktiven Umgebung arbeitet. Die Laborumgebung besteht aus einer VMware ESX-Virtualisierungsumgebung mit mehreren physischen Rechnern, Switches und einem NAS. Der Aufbau dient zur Überprüfung der Anforderungskriterien von Abschnitt 3.2.

4.2.1 Hardwarekomponenten

Als zentrales NAS ist eine TS-819 der Firma QNAP mit 12 x 2 TB Seagate Festplatten im Einsatz. Das Speichersystem selbst ist mit einem Intel Dual Core Prozessor mit einem Gigabyte RAM Hauptspeicher ausgestattet. Als Schnittstellen zur Außenwelt stehen zwei Ein-Gigabit-Ports für Ethernet und sechs USB-Ports zur Verfügung.

In der ESX-Umgebung werden vier Optiplex-Workstations der Firma DELL als physische Plattform für die virtuellen Maschinen eingesetzt. Drei Workstations stammen aus der

[21] Domain Name System (DNS) ist ein Dienst, um die numerischen IP-Adressen in Namen aufzulösen.

[22] Network Time Protocol (NTP) handelt es sich um ein Protokoll, um Uhrzeiten über ein Netzwerk zu synchronisieren.

Modellreihe 780 und eine Workstation aus der Modellreihe 7010. In der nachfolgenden Tabelle ist die Ausstattung der Modelle gelistet.

Hersteller	Modell	CPU-Typ	Taktrate	Sockel	Cores	Logical CPU	RAM
Dell	780	E8400	3 GHz	1	2	4	16 GB
Dell	7010	I7-3770	3,4 GHz	1	4	8	8 GB

Tabelle 5: ESX-Server Ausstattung

Die einzelnen Komponenten werden über zwei Gigabit-Ethernet-Switches verbunden, die jeweils mit fünf Ports ausgestattet waren.

4.2.2 Physische Verkabelung

Die ESX-Maschinen waren über einen Ethernet-Switch direkt mit dem NAS-System verbunden und über den zweiten Switch mit dem Company-LAN, um die Dienste DNS und NTP beziehen zu können. Für die Verwaltung der ESX-Umgebung wurde eine virtuelle Maschine eingesetzt, die eine dediziert physische Maschine zugewiesen war. Durch dieses Vorgehen konnte die Verwaltungssoftware der ESX-Umgebung die anderen ESX-Server nicht beeinflussen. Die physische Maschine, die für die Verwaltung eingesetzt wurde, ist mit einem Ein-Gigabit-Netzwerk-Port ausgerüstet. Der Netzwerk-Port wurde für die Verbindung in das Company-LAN und für die Verwaltung der ESX-Umgebung eingesetzt. In der Abbildung 20 wird die physische Verkabelung grafisch dargestellt.

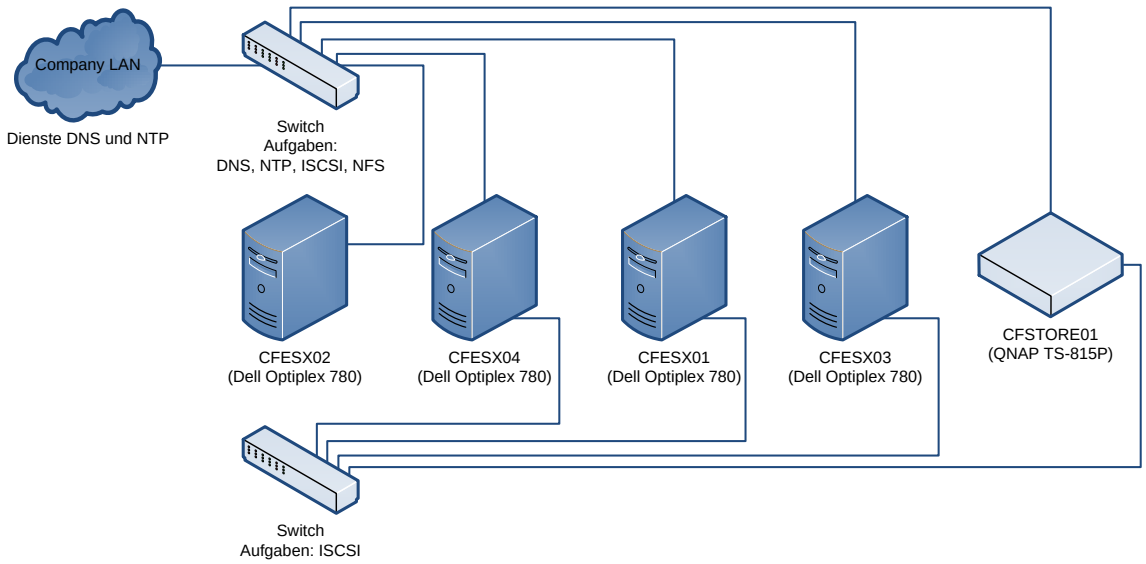


Abbildung 20: Aufbau Praxisumgebung

Das NAS-System war wie die restlichen drei physischen Maschinen, mit zwei Ein-Gigabit-Netzwerk-Ports ausgerüstet. Eines davon wurde verwendet, um eine direkte Verbindung über einen der Ethernet-Switches zu den ESX-Servern herzustellen, das zweite Netzwerk-Port, um die Dienste NTP und DNS über das Company-LAN zu beziehen.

Die Maschinen CFESX01 und CFESX03 wurden als Plattform für die virtuellen Maschinen eingesetzt. Die Maschine CFESX04 betreibt die SDFS-Appliance und CFESX02 die Verwaltungssoftware für die ESX-Umgebung. Der Ethernet-Switch im unteren Bereich der Abbildung 20 wurde für den ISCSI-Datenverkehr zwischen CFSTORE01 und den ESX-Rechnern CFESX01 und CFESX03 verwendet. Der obere Ethernet-Switch war für die Verwaltung, die Verbindung in das Company-LAN und den NFS-Datenverkehr zwischen CFESX04 und CFESX01 und CFESX03 zuständig.

4.2.3 Software

Als Virtualisierungssoftware kam das Produkt ESX in der Version 5.0 von der Firma VMware zum Einsatz. Um eine vollwertige ESX-Umgebung zu erhalten, wurden die VMware-Komponenten VMware VSphere[23] und der VMware VCenter Server[24] zusätzlich eingesetzt. Beide Produkte hatten die Versionsnummer 5.0.0. Nur durch den Einsatz dieser Komponenten können die Möglichkeiten einer ESX-Umgebung voll ausgeschöpft werden.

Als DEDUP-System kommt das freie Software-Projekt SDFS zum Einsatz. Die Software-Appliance in der Version 1.5.0 wurde eingesetzt. Das SDFS-Modul innerhalb der Appliance erhielt ein Update auf die Version 1.8.0.

Das Linux-Betriebssystem Red Hat in der Version 6.2 war die Basis für die virtuellen Maschinen, die für die Tests in der Praxisumgebung eingesetzt wurden.

Das NAS-System der Firma Qnap hatte die Firmware-Version 3.6.0 Build 0210T installiert.

4.2.4 Konfiguration

4.2.4.1 NAS

Das NAS-System CFSTORE01 war mit insgesamt acht Festplatten vom Hersteller Seagate (Modellnummer ST32000542AS) zu je zwei Terabyte Speicherkapazität ausgestattet.

[23] Bei VMware VSphere handelt es sich um eine grafische Oberfläche, die es ermöglicht, ESX-Umgebungen zu verwalten.

[24] VMware VCenter Server wird benötigt, wenn eine zentrale Verwaltung von mehreren ESX-Servern gewünscht wird.

Diese Festplatten wurden mit einem RAID[25] 10 zusammengefasst, um einen hohen Datendurchsatz und Datensicherheit zu erhalten.

In diesem RAID-System wurden drei ISCSI-Volumes angelegt, die als *datastores* der ESX-Umgebung dienten. Das Volume Q_ISCSI_SDFS_VOL01 war als *datastore* dediziert für die SDFS-Appliance im Einsatz. Die Volumes Q_ISCS_VOL02 und Q_ISCS_VOL03 waren als *datastores* für die diversen virtuellen Maschinen-Instanzen reserviert.

4.2.4.2 ESX-Umgebung [VMD2012]

Der Funktionsumfang der VMware-ESX-Umgebung entsprach dem produktiven System vom Kunden, die Leistungsfähigkeit war durch die eingesetzte Hardware begrenzt.

Die ESX-Umgebung bestand aus vier ESX-Servern, die mit dem VMware VCenter-Server verwaltet wurden. Der ESX-Server CFESX02 wurde dediziert für die Verwaltungssoftware der ESX-Umgebung ausgewählt. Diese Maschine besaß nur ein Netzwerk-Port. Als *datastore* mit dem Namen L_CFESX02 wurde die lokale Festplatte im ESX-Server CFESX02 verwendet, da bei dieser Maschine keine ISCSI-Anbindung benötigt wurde. Abbildung 21 zeigt den Aufbau der ESX-Umgebung und die zugeteilten Aufgaben der jeweiligen Komponenten.

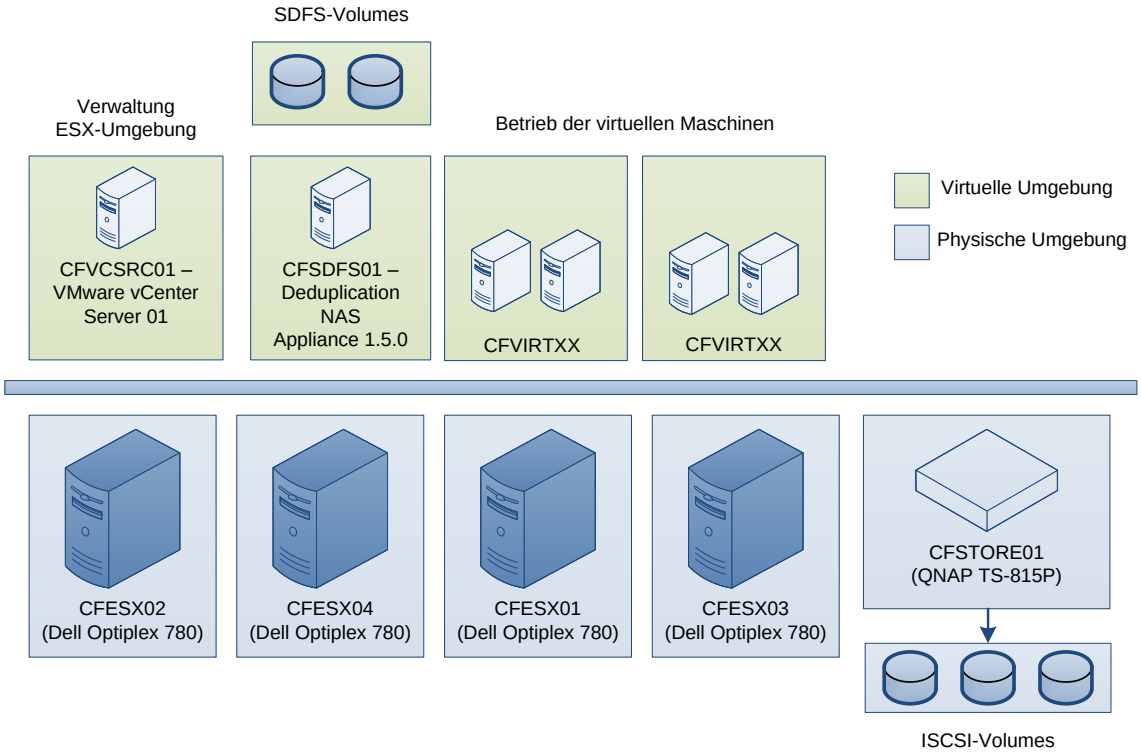


Abbildung 21: ESX-Umgebung Aufbau

[25] Redundant Array of Independent Disks (RAID) ist ein logischer Verbund von Festplatten um eine höhere Verfügbarkeit bei Ausfall von physischen Festplatten zu erreichen. Bei RAID 10 handelt sich um eine Festplattenkombination, die mindestens vier Festplatten voraussetzt.

Die Maschinen CFESX01, CFESX03 und CFESX04 waren mit zwei Netzwerk-Ports ausgerüstet. Ein Netzwerk-Port wurde ausschließlich für die ISCSI-Kommunikation zwischen dem ESX-Server und dem NAS-System (CFSTORE01) vorgesehen, der zweite Netzwerk-Port wurde für die Verwaltung, den NFS-Datenverkehr und den ISCSI-Datenverkehr zwischen ESX-Server und der SDFS-Appliance (CFSDFS01) verwendet. Der ESX-Server CFESX01 und CFESX03 wurde für den Betrieb der virtuellen Maschinen verwendet. CFESX04 war exklusive für die SDFS-Software-Appliance reserviert.

Für die ISCSI-Kommunikation zwischen ESX-Server, SDFS-Appliance und NAS-System musste ein ISCSI-Adapter konfiguriert werden. Es wurde der Software-ISCSI-Adapter von VMware, der mit dem Produkt ESX mitgeliefert wird, eingesetzt. Dieser ISCSI-Adapter stellte die Festplatten von der SDFS-Appliance und dem NAS-System der ESX-Umgebung zu Verfügung.

4.2.4.3 Virtuelle Maschinen

Virtuelle Maschinen dienten als Datenbasis für die Überprüfung der SDFS-Appliance. Sie wurden eingesetzt, um die DEDUP-Ratio, Performance und die Stabilität der SDFS-Appliance zu analysieren. Als Storage-System für diese Maschine wurde je nach Szenario das NAS-System CFSTORE01 oder die SDFS-Appliance CFSDFS01 verwendet.

Als virtualisierte Hardware für die Maschineninstallation wurde immer die gleiche ESX-Hardware-Vorlage verwendet. Tabelle 6 zeigt die Eckpunkte der virtuellen Maschine.

Hersteller	VM Version	CPU	Festplatte	Sockel	Cores	RAM
ESX	8	1	100 GB	1	1	2 GB

Tabelle 6: Virtuelle Hardware

Als Gast-Betriebssystem wurde das Linux-Betriebssystem Red Hat Enterprise Linux 6.2 (64-Bit) definiert. Dieses Betriebssystem wird als Basis für eine Vielzahl von IT-Dienstleistungen beim Kunden eingesetzt.

„Thin Provision[26]“ wurde als Typ für die Festplatte der virtuellen Maschine von ESX festgelegt. Detaillierte Konfigurationseinstellungen sind unter Anlage 4 zu finden. In dieser Anlage ist die vollständige ESX-Konfigurationsdatei gelistet.

4.2.4.4 Netzwerkkonfiguration

Der ESX-Umgebung standen zwei TCP/IP-Netzwerkbereiche zur Verfügung. Um die einzelnen Komponenten der Laborumgebung über das Company-LAN zu erreichen, wurde das Netz 192.168.7.0/24 verwendet. Das TCP/IP-Netz 10.0.0.0/24 war für die Verbindung

[26] Thin Provision ist ein Verfahren, um Speicherkapazitäten virtuellen Maschinen bereitzustellen (siehe https://www.vmware.com/products/datacenter-virtualization/vsphere/storage-thin-provisioning.html).

zwischen dem NAS-System und dem ESX-Server zuständig. Dieses LAN wurde ausschließlich in der Laborumgebung verwendet und war völlig vom Company-LAN getrennt.

Die interne ESX-Bezeichnung *„VMStorage“* wurde für das LAN 10.0.0.0/24 verwendet. Hauptsächlich erfolgte in diesem LAN der Datentransfer zwischen dem NAS-System (CFSTORE01) und den ESX-Servern. Das Netz 192.168.7.0/24 hatte den Namen *„VM Network“* und wurde für das Management der ESX-Umgebung und den Datenaustausch zwischen ESX-Servern und der SDFS-Appliance eingesetzt.

Zwischen dem LAN *„VM Network“* und *„VMStorage“* gab es keine physische Verbindung. Zusätzlich trennte die ESX-Konfiguration die Netze logisch voneinander. Virtuelle Maschinen hatten keinen Zugriff auf das LAN *„VMStorage“* und konnten nur eine Verbindung zu dem LAN *„VM Network“* aufnehmen.

Die ESX-Server nutzten das LAN *„VMStorage“* für das Protokoll ISCSI. ISCSI-Target war das NAS-System CFSTORE01, der ISCSI-Initiator waren die ESX-Server CFESX01, CFESX03 und CFESX04.

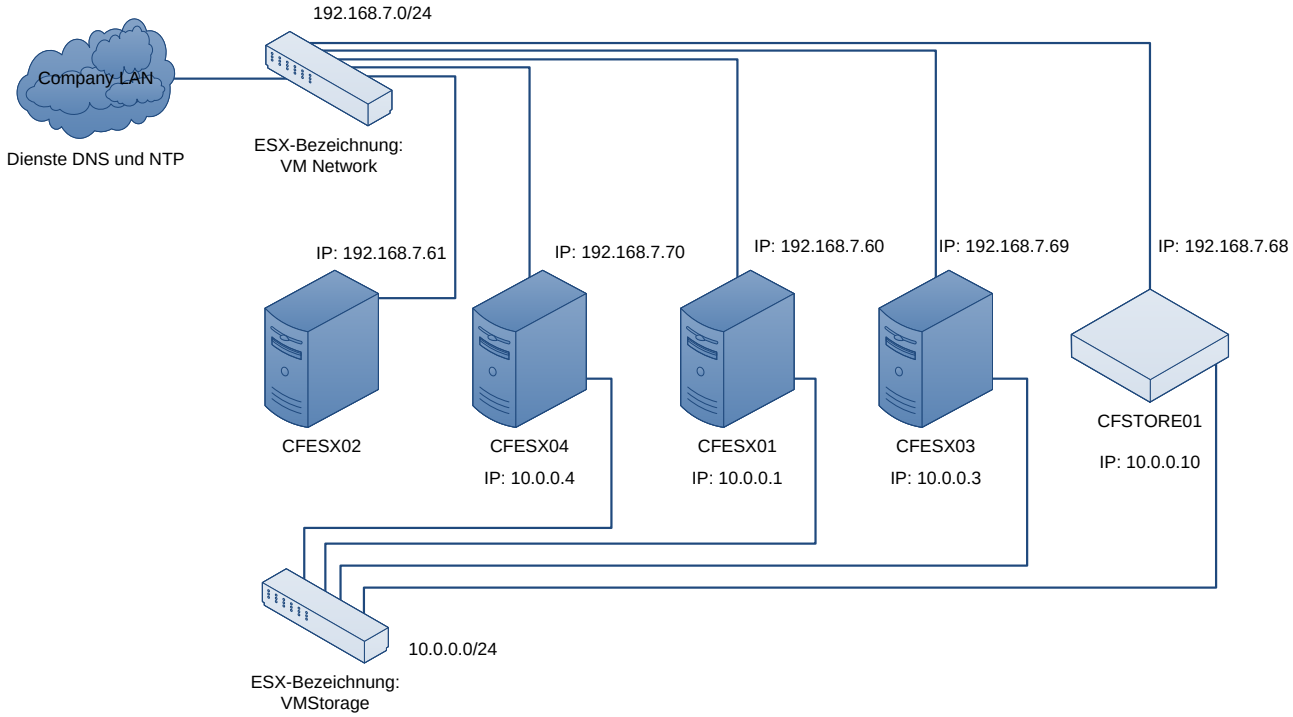


Abbildung 22: IP-Adressen Laborumgebung

Das NAS-System (CFSTORE01) wurde mit jeweils einem Netzwerk-Port in das Netz 192.168.7.0/24 und 10.0.0.0/24 verbunden.

4.2.4.5 Storage-Konfiguration [QNA2012]

Die acht Festplatten vom NAS-System wurden in einem RAID10-Verbund zusammengeschaltet. Die Gesamtkapazität belief sich auf 7,3 TB. In diesem RAID wurden mit der Konfigurationsoberfläche des Herstellers drei ISCSI-Targets erstellt.

Das ISCSI-Target vol01 war als *datastore* für die SDFS-Appliance reserviert. Für die virtuellen Maschinen wurden zwei weitere ISCSI-Targets vol02 und vol03 erstellt. Die ISCSI-Targets wurden mit dem ISCSI-Software-Adapter, der bei ESX mitgeliefert wird, eingebunden. In der nachfolgenden Tabelle ist die Verknüpfung zwischen ISCSI-Targets auf dem NAS-System mit der ESX-Umgebung ersichtlich.

CFSTORE01	CFSTORE01-LUN	Kapazität	*datastore name*
iqn.2004-04.com.qnap:ts-809:iscsi.esx.c06bb1	vol01	1000 GB	Q_ISCSI_SDFS_VOL01
iqn.2004-04.com.qnap:ts-809:iscsi.esx.c06bb1	vol02	1000 GB	Q_ISCSI_VIRT_VOL02
iqn.2004-04.com.qnap:ts-809:iscsi.esx.c06bb1	vol03	1000 GB	Q_ISCSI_VIRT_VOL03

Tabelle 7: ISCSI-Targets

In der Tabelle 7 werden die Namen der ISCSI-Targets und den dazugehörigen *datastores* in der ESX-Umgebung gelistet.

4.3 Installation SDFS

Um den Installationsaufwand der SDFS-Software gering zu halten, kam die vorkonfigurierte virtuelle Appliance vom Entwickler Mark Silverberg zum Einsatz. Diese Appliance verhält sich nach außen wie ein NAS-System, das über die Protokolle ISCSI und NFS SDFS-Volumes anbietet. Die Software-Appliance kann im Format OVF[27] von der Projekthomepagehttp://www.opendedup.com bezogen werden.

4.3.1 Deployment

Die Maschine wird mithilfe der VSphere-Software in die ESX-Umgebung *deployed*. OVF wird von VMware vollständig unterstützt und benötigt keine Software von Drittherstellern um diese zu verarbeiten. Als physischer Rechner wurde die Maschine CFESX04 eingesetzt, da diese die Anforderung der virtuellen Maschine erfüllt.

[27] Open Virtualization Format (OVF) ist ein offener Standard, der es erlaubt, virtuelle Maschinen zwischen den diversen Anbietern zu verteilen.

Abbildung 23 zeigt die verwendeten Ressourcen nach dem *deploy* der Appliance.

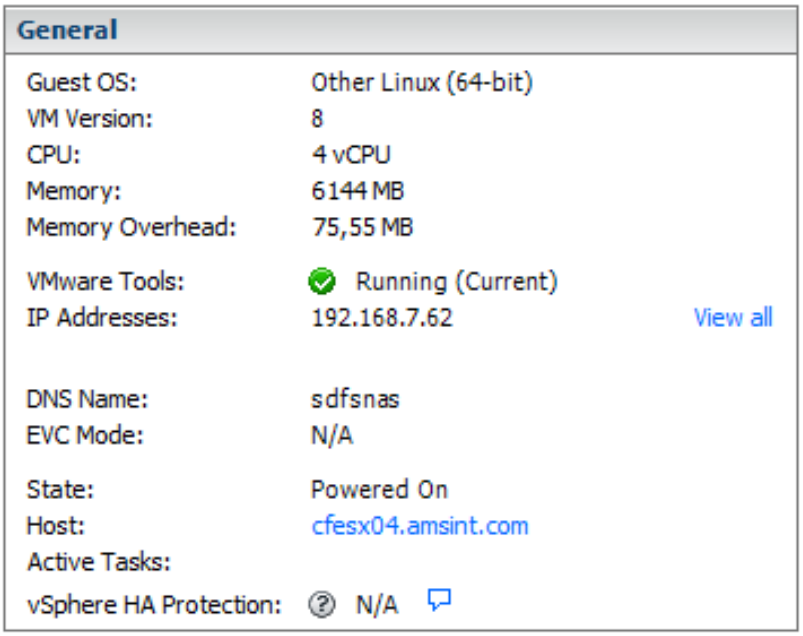


Abbildung 23: VM-Information SDFS-Appliance

4.3.2 Erstkonfiguration

Nach dem Bereitstellen der Maschine in der ESX-Umgebung wurde sie nach der Anleitung auf der Projekthomepage konfiguriert. [OPENG2012]

Einige Einstellungen mussten nach dem *deploy* der SDFS-Appliance geändert werden. Die Netzwerkkonfiguration wurde von DHCP auf eine statische Netzwerkkonfiguration geändert. Zusätzlich wurde die Konfigurationsdatei */etc/network/interfaces* mit den Optionen *dns-nameservers* und *dns-search* erweitert. Diese Erweiterung war notwendig, um eine funktionierende Namensauflösung zu erhalten.

Die Zeitzone musste zusätzlich innerhalb der SDFS-Appliance geändert werden, um gültige Zeitstempel in den Logdateien zu erhalten. Diese Einstellung war über die SDFS-Management-Konsole nicht möglich. Mit dem Linux-Befehl *dpkg-reconfigure tzdata* konnte die Zeitzone in der Kommandozeile eingerichtet werden.

4.3.3 Bereitstellen eines SDFS-Volumes

Die SDFS-Management-Konsole ermöglicht es, SDFS-Volumes zu erstellen, zu verwalten und zu löschen. Nach der Einrichtung eines Volumes kann dieses über ISCSI oder NFS in der ESX-Umgebung registriert werden. Dazu müssen die Account-Daten der ESX-Umgebung im Punkt Administration im SDFS-Webinterface eingegeben werden. Im folgenden Abschnitt wird ein Volume im SDFS-Webinterface erstellt, das für die Deduplizierung von virtuellen Maschinen geeignet ist.

Bei der Erstellung des SDFS-Volumes ist es möglich, zwei Kapazitäten anzugeben. Der erste Wert entspricht der Speicherkapazität, die bei der ESX-Umgebung angezeigt wird.

Es handelt sich um den Duplikatsspeicher. Der zweite Wert ist die Speicherkapazität, die dem SDFS-Volume innerhalb der SDFS-Appliance zur Verfügung steht (Unikatsspeicher). Diese Einstellungen erlauben es, SDFS-Volumes zu überzeichnen, indem ein SDFS-Volume mit 500 GB Duplikatsspeicher und 100 GB Unikatsspeicher konfiguriert wird. Durch diese Einstellung muss SDFS eine DEDUP-Ratio von mindestens 5:1 erreichen. Fällt die DEDUP-Ratio unter diesen Wert, kann der Unikatsspeicher keine Daten mehr aufnehmen, obwohl der Duplikatsspeicher freien Speicherplatz ausweist.

Die nachfolgende Abbildung 24 zeigt die Konfiguration des SDFS-Volumes mit dem Namen *cfsfds_4k_vol01* über das Webinterface. Die Konfigurationseinstellungen wurden so gewählt, wie sie beim Kunden derzeit im Einsatz sind. Die 4KB-Blocksize wurde gewählt, da diese laut [OPENA2012] die beste DEDUP-Ratio für virtuelle Maschinen bietet. Eine 5:1-Datenreduktion wurde mindestens angenommen, deswegen wurde der Unikatsspeicher mit 100 GB festgelegt.

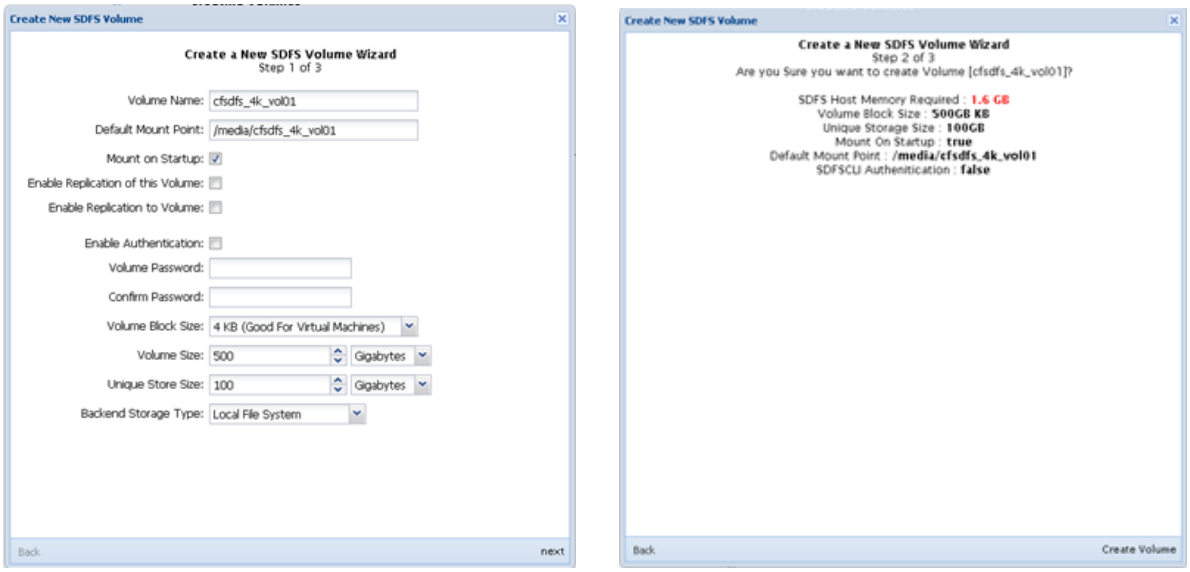


Abbildung 24: SDFS-Volume-Erstellung

Das Webinterface kann ein erstelltes Volume überwachen und Informationen im Steckbrief Format oder grafisch in einem Kuchendiagramm anzeigen. (Abbildung 25). Als Default-Einstellung wird bei SDFS DEDUP-Inband verwendet. DEDUP-Outband kann nicht über das Webinterface eingestellt werden. Für DEDUP-Outband[28] muss die XML-Konfigurationsdatei per Hand geändert werden.

[28] DEDUP-Outband wird bei SDFS unter dem Begriff *batch based* geführt (siehe http://www.opendedup.org/administrators-guide#inline).

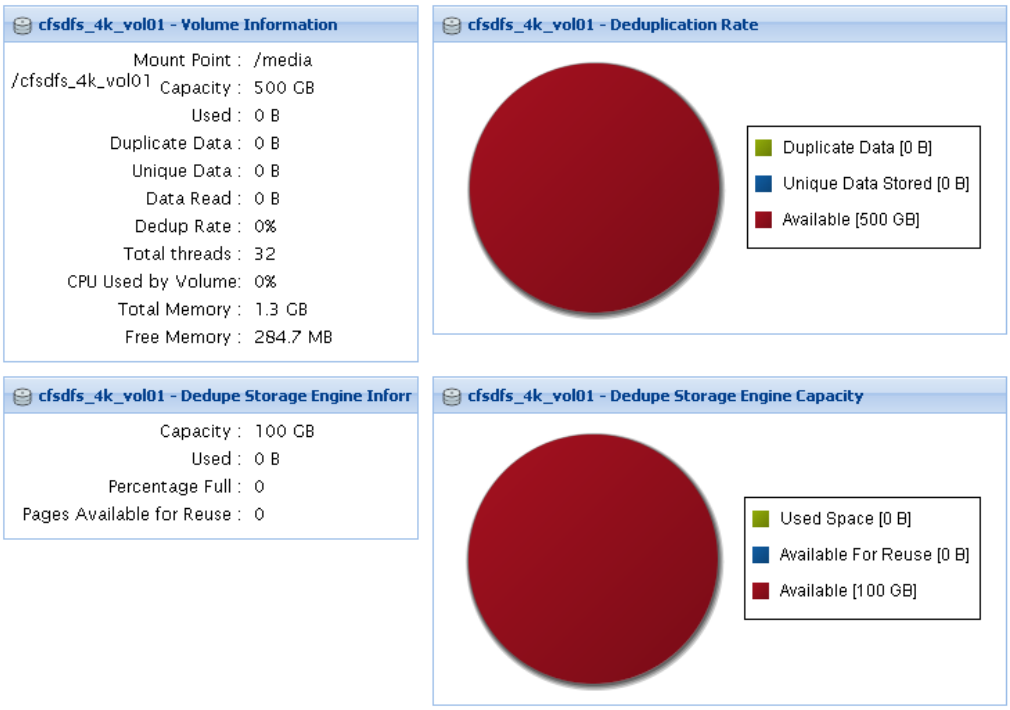


Abbildung 25: SDFS-Volume-Information

Die Konfiguration eines SDFS-Volume wird in einer XML[29]-Datei im Ordner /etc/sdfs innerhalb der SDFS-Appliance gespeichert. In Anlage 1 ist die Konfiguration eines SDFS-Volume gelistet. Im laufenden Betrieb bietet die SDFS-Management-Konsole noch zusätzlich die Funktion einer Echtzeitüberwachung. Über diese Anzeige werden die aktuellen verarbeiteten Daten getrennt in Unikate und Duplikate angezeigt. Abbildung 26 zeigt die Echtzeitüberwachung.

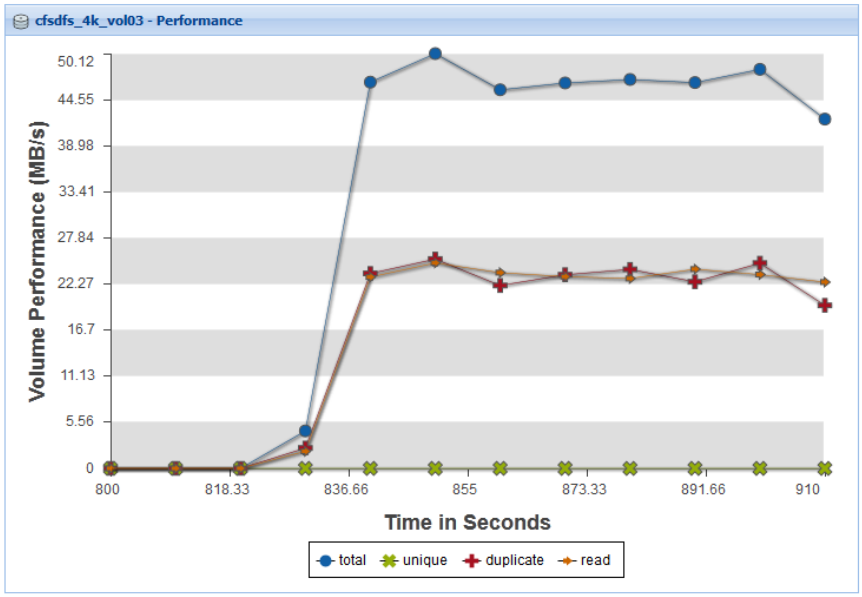


Abbildung 26: SDFS-Echtzeitüberwachung

[29] XML ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien (siehe http://de.wikipedia.org/wiki/Extensible_Markup_Language).

4.4 SDFS-Datenreduktion

In diesem Abschnitt wird das Anforderungskriterium Datenreduktion über die DEDUP-Ratio ermittelt. Als Datenbasis werden virtuelle Maschinen mit dem Linux-Betriebssystem Red Hat eingesetzt. Ein SDFS-Volume dient als *datastore* für die virtuellen Maschinen. Die Konfiguration des SDFS-Volumes wird, wie in Abschnitt 4.3.3 beschrieben, vorgenommen.

SDFS-Volumes werden als NFS-SDFS-Volumes bezeichnet, wenn sie über NFS angebunden werden, ISCSI-SDFS-Volume, wenn ISCSI anstatt NFS verwendet wird. ISCSI-Volumes vom NAS-System CFSTORE01 werden als native ISCSI bezeichnet.

Die DEDUP-Ratio wurde in drei Szenarien überprüft, die in der täglichen Arbeit mit einer ESX-Umgebung auftreten können.

Szenario	Beschreibung
Standard-Installation	Im ersten Szenario wurden vier Maschinen mit dem Installationsprogramm, das vom Hersteller geliefert wird, ident installiert. Dieses Szenario soll Situationen wiedergeben, in denen ein Administrator eine virtuelle Maschine manuell aufsetzt.
Migration virtueller Maschinen	Das zweite Szenario vergleicht die DEDUP-Ratio mit dem ersten Szenario, wenn die virtuellen Maschinen auf einen neuen *datastore* verschoben wurden. Während des Betriebs können immer wieder Situationen entstehen, die es erfordern, Maschinen von einem *datastore* auf einen neuen *datastore* zu verschieben. Bespielhafte Gründe sind Wartungsarbeiten am Storage, Performanceprobleme oder auch zu geringer Speicherplatz.
ESX-Template-Installation	Wie sich die DEDUP-Ratio bei einer ESX-Template[30]-Maschineninstallation verhält, wurde im letzten Szenario überprüft. Diese Art der Installation wird oft in einer ESX-Umgebung eingesetzt, um die manuelle Installation nicht durchführen zu müssen. Die manuelle Installation ist im Gegensatz zu der Template-Installation viel zeitintensiver und durch das manuelle Arbeiten fehleranfälliger.

[30] Bei ESX-Template handelt es sich um einen Abzug einer vorkonfigurierten virtuellen Maschine, die als Vorlage für weitere virtuellen Maschinen dient.

Nach jedem Szenario, wenn nicht anders angegeben, wurde ein neues SDFS-Volume verwendet, um die gleiche Ausgangsbasis zu erhalten. Bestehende SDFS-Volumes wurden gelöscht oder deaktiviert, um eine Beeinflussung zu verhindern. Jedes Szenario wurde zweimal durchgeführt, um die ermittelten Werte zu verifizieren.

4.4.1 Standard-Installation

Bei der ersten Überprüfung der Leistungsfähigkeit von SDFS wurde die DEDUP-Ratio ermittelt. Als Datenbasis wurde ein Red-Hat-Linux-Betriebssystem ausgewählt, das beim Kunden häufig zum Einsatz kommt.

Das Red Hat-Betriebssystem wurde viermal auf einem NFS-SDFS-Volume mit dem Namen S_NFS_VOL01 installiert, das über NFS an die ESX-Umgebung angebunden war. Nach jeder Installation wurde der verbrauchte Speicherplatz protokolliert. Die virtuellen Maschinen erhielten die Namen „*cfvirt01 – RH5 – Standard-Installation*“ bis „*cfvirt04 – RH5 – Standard-Installation*“. Der verbrauchte Speicherplatz setzt sich aus den Unikaten und der Verwaltung für diese Unikate zusammen.

Die Werte wurden nicht über das Web-interface ermittelt, sondern direkt innerhalb der SDFS-Appliance mit dem Linux-Befehl *du* im Verzeichnis */opt/sdfs*. In Anlage 9 ist das Vorgehen protokolliert. Dieser Weg musste gewählt werden, da das Webinterface die Verwaltung nicht einbezieht und dadurch die Ergebnisse verfälscht. Der Platzverbrauch wurde dem Speicherplatzbedarf der virtuellen Maschine gegenübergestellt. Nach jeder virtuellen Maschineninstallation wurde ein *sdfscli --cleanstore* innerhalb der SDFS-Appliance durchgeführt, um die *reusable pages* in die DEDUP-Ratio miteinzubeziehen.

Bei den *reusable pages* handelt es sich um Speicherplatz, der beim Löschen von Daten in einem SDFS-Volume auftritt. Dieser Speicherplatz wird von SDFS nicht mehr freigegeben, sondern für eine Wiederverwendung reserviert.

Die virtuelle Maschine wurde mit dem Linux-Betriebssystem Red Hat in der Version 6.2 in der 64-Bit-Variante aufgesetzt. Die Installation wurde mit dem Installations-Wizard, der vom Hersteller bereitgestellt wird, durchgeführt. Bis auf den Punkt Paketauswahl fußt die Installation auf den Default-Einstellungen des Herstellers. Mit dem Softwareprojekt KDE[31] wurde diese erweitert, da KDE beim Kunden zusätzlich eingesetzt wird. Die Installationskonfiguration der virtuellen Maschine befindet sich in Anlage 3. Die virtuellen Maschinen wurden auf dem ESX-Server CFESX01 betrieben.

In Tabelle 8 sind die ermittelten Werte ersichtlich. In der Spalte DEDUP-Ratio wird in Klammer der DEDUP-Ratio-Wert abzüglich der *reusable pages* angegeben.

[31] Bei K Desktop Environment handelt es sich um eine grafische Benutzeroberfläche, die vor allem im europäischen Raum eingesetzt wird.

OS	Anzahl	BS	LNX (KB)	Reusable Pages	SDFS	DEDUP-Ratio
Red Hat 6.2	1	4 KB	3839899	233989	4987788	0,77:1 (0,95:1)
Red Hat 6.2	2	4 KB	7679798	211153	5449272	1,41:1 (1,67:1)
Red Hat 6.2	3	4 KB	11519697	214926	5904096	1,95:1 (2,28:1)
Red Hat 6.2	4	4 KB	15359596	199387	6367892	2,41:1 (2,76:1)

Tabelle 8: Standard-Installation

Die theoretische Datenreduktion in diesem Szenario war 4:1, wenn die Verwaltung außer Acht gelassen wurde. Eine DEDUP-Ratio von 2,76:1 konnte nach dem Abzug der *reusable pages* erreicht werden. Mit den *reusable pages,* was dem tatsächlichen Speicherplatzverbrauch innerhalb der SDFS-Appliance entspricht, wurde eine DEDUP-Ratio von 2,41:1 erreicht. Dieser DEDUP-Ratio-Wert lag deutlich unter dem theoretisch möglichen.

4.4.2 Migration virtuelle Maschinen

Migrationen von virtuellen Maschinen auf einen neuen *datastore* sind in einer ESX-Umgebung eine häufige Aufgabe. Dieses Szenario stellt die Situation mit den vier virtuellen Maschinen, die in Abschnitt 4.4.1 erstellt wurden, nach. Nacheinander wurde jede der vier virtuellen Maschinen auf einen neuen *datastore* verschoben. Der neue *datastore* ist wieder ein NFS-SDFS-Volume (S_NFS_VOL02). Die virtuellen Maschinen wurden vom *datastore* S_NFS_VOL01 aus Abschnitt 4.4.1auf den neuen *datastore* S_NFS_VOL02 migriert.

Die Migration der Maschinen wurde mit der ESX-Funktion *migrate datastore* durchgeführt. Nach jeder Migration wurde der benötigte Speicherplatz innerhalb der SDFS-Appliance protokolliert. In der folgenden Tabelle 9 sind die Werte ersichtlich.

OS	Durch-lauf	BS	LNX (KB)	Reusable Pages	SDFS	DEDUP-Ratio
Red Hat 6.2	1	4 KB	3839899	10376	4037636	0,95:1 (0,95:1)
Red Hat 6.2	2	4 KB	7679798	11244	4582404	1,68:1 (1,68:1)
Red Hat 6.2	3	4 KB	11519697	11448	5011388	2,30:1 (2,30:1)
Red Hat 6.2	4	4 KB	15359596	10284	5526564	2,78:1 (2,78:1)

Tabelle 9: Migration virtuelle Maschinen

Die auffälligste Veränderung ist bei den *reusable pages* zu beobachten. Im Vergleich mit dem vorhergehenden Abschnitt ist der Wert um ein Vielfaches niedriger. Die DEDUP-Ratio stieg nicht signifikant an.

4.4.3 Template-Installation

Die DEDUP-Ratio blieb bei einer klassischen Installation weit hinter dem theoretischen Wert zurück. In diesem Abschnitt wurden die virtuellen Maschinen über ein ESX-Template angelegt. Diese Art der Maschineninstallation ist die häufigste eingesetzte in einer ESX-Umgebung. Standardkonfigurationen, die für die Integration in eine IT-Umgebung nötig sind, können in einem Template berücksichtigt werden. Dadurch wird der Zeitbedarf von der Installation bis zur Inbetriebnahme der virtuellen Maschine verkürzt.

Als Vorlage für das ESX-Template diente die erste virtuelle Maschine „*cfvirt01 – RH5 – Standard-Installation*“ aus Abschnitt 4.4.1. Die ESX-Funktion „*Convert to Template*“ wurde für Erstellung des Templates benutzt. Mit diesem ESX-Template wurde das Szenario einer Template-Installation dargestellt. Anstatt eine komplette Installation manuell durchzuführen, wurden vom Template Abzüge generiert.

Wie bei der ersten und zweiten DEDUP-Ratio-Überprüfung wurde ein neuer *datastore* mit dem Namen S_NFS_VOL03 angelegt, das hintereinander mit vier neuen virtuellen Maschinen aus dem ESX-Template beschrieben wurde. Nach jeder Maschinenerstellung wurden die ermittelten Werte innerhalb der SDFS-Appliance in Tabelle 10 eingetragen.

OS	Anzahl	BS	LNX (KB)	Reusable Pages	SDFS	DEDUP-Ratio
Red Hat 6.2	1	4K B	3839899	24	4011820	1,96:1 (1,96:1)
Red Hat 6.2	2	4K B	7679798	24	4039512	1,90:1 (1,90:1)
Red Hat 6.2	3	4K B	11519697	26	4067180	2,83:1 (2,83:1)
Red Hat 6.2	4	4K B	15359596	27	4094796	3,75:1 (3,75:1)

Tabelle 10: ESX-Template-Installation

Die DEDUP-Ratio wurde durch die Template-Installation erheblich verbessert. Zusätzlich gingen die *reusable pages* auf einen verschwindend kleinen Wert zurück und beeinflussten die DEDUP-Ratio nicht merkbar. Nach der Speicherung von vier virtuellen Maschinen konnte eine DEDUP-Ratio von 3:75:1 erreicht werden.

4.4.4 Zusammenfassung DEDUP-Ratio

Im ersten Testszenario, das sich mit der manuellen Installation von virtuellen Maschinen beschäftigt, konnte eine DEDUP-Ratio von 2,76:1 erreicht werden. Dieser Wert lag deutlich unter dem theoretischen Wert von 4:1. Obwohl viermal eine Maschine mit dem gleichem Installationsmedium und gleichen Einstellungen aufgesetzt wurde, konnte SDFS nur relativ wenige Duplikate erkennen.

Die hohe Anzahl der *reusable pages* lässt sich mit der Installationsart von Red Hat erklären. Das Betriebssystem Red Hat besteht nicht aus einem großen monolithischen Block, sondern aus vielen kleinen Paketen, die einzeln kopiert, entpackt, installiert und konfiguriert werden. Diese Art der Installation erzeugt temporäre Daten, die nach der Installation

wieder gelöscht werden. Dieser Prozess führt schlussendlich zu den *reusable pages* im SDFS-Volume.

Die Migration der vier virtuellen Maschinen auf ein neues SDFS-Volume verbesserte die DEDUP-Ratio nur leicht. Deutlich wurde der Wert der *reusable pages* beeinflusst. Dadurch, dass die virtuellen Maschinen schon installiert waren, mussten keine temporären Daten, wie bei der manuellen Installation, angelegt werden. Die DEDUP-Ratio wurde durch den geringeren Verwaltungsaufwand positiv beeinflusst.

Bei der Template-Installation wurde die höchste DEDUP-Ratio von 3,75:1 erreicht. Sie reichte an den theoretischen Wert von 4:1 heran. Dieser hohe Wert kann mit der Funktion eines ESX-Templates erklärt werden. Wird eine virtuelle Maschine aus einem ESX-Template erstellt, wird nicht nur der Inhalt der Maschine übernommen, sondern auch die Reihenfolge, wie die Daten abgelegt wurden. Das ESX-Template wird sequentiell gelesen und geschrieben. Dadurch war es SDFS möglich, mehr Duplikate zu finden. In Abbildung 27 werden die gesammelten Ergebnisse von diesem Abschnitt zusammenfassend gezeigt. Die *reusable pages* wurden mit dem Begriff *rpages* Abgekürzt.

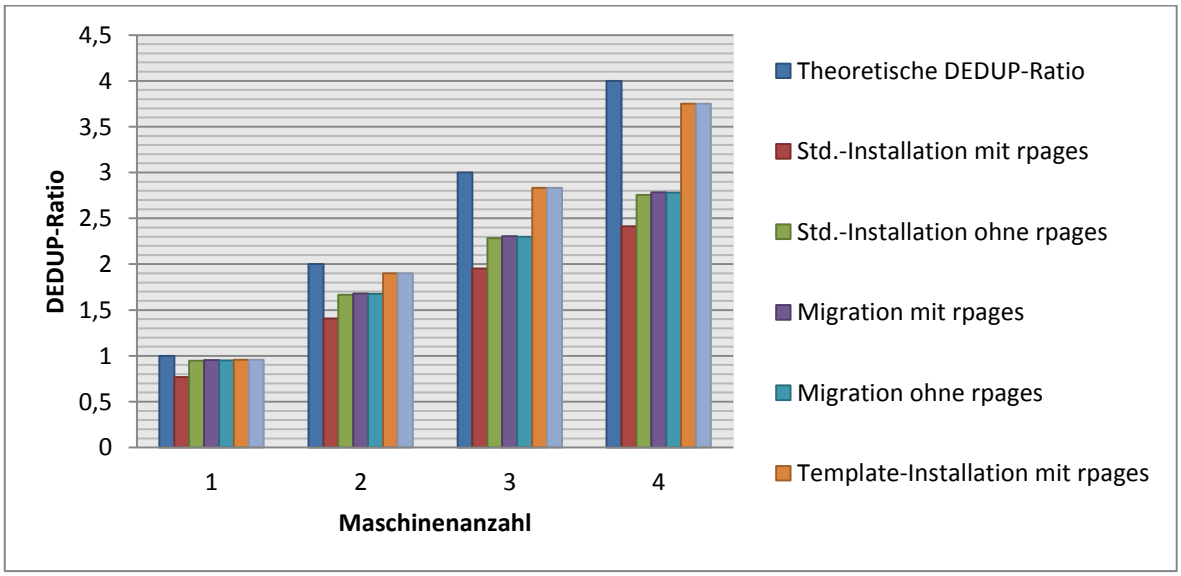


Abbildung 27: DEDUP-Ratio Zusammenfassung

Abschließend kann die Empfehlung für eine ESX-Template-Installation gegeben werden. SDFS erreichte die höchste DEDUP-Ratio mit dieser Art einer Maschineninstallation.

4.5 SDFS-Performance

Dieser Abschnitt beschäftigt sich mit dem Anforderungskriterium Performance. Ein ISCSI-SDFS-Volume der SDFS-Appliance wird einem native ISCSI-Volume vom NAS-System gegenübergestellt. Um die Art der *datastores* namentlich zu unterscheiden wurde folgende Namenskonvention verwendet. ISCSI-SDFS-Volumes erhielten den Anfangsbuchstaben „S“, native ISCSI-Volumes den Anfangsbuchstaben „Q“ gefolgt vom verwendeten Datenübertragungsprotokoll und einer fortlaufenden Nummer, wenn mehrere benötigt wurden.

Mehrere ESX-typische Aufgaben wurden zuerst auf einem native ISCSI-Volume und danach auf einem ISCSI-SDFS-Volume durchgeführt. Die benötigte Zeit der Aufgaben wurde protokolliert und verglichen. Zusätzlich wird im Abschnitt 4.5.2 die Entlastung eines Storage-Systems durch die Datendeduplizierung gezeigt. Jede Überprüfung wurde zweimal durchgeführt um die gemessenen Werte zu verifizieren. [BENC2009]

4.5.1 ESX-Aufgaben

In diesem Abschnitt werden Geschwindigkeitsmessungen bei drei typischen Arbeiten in einer ESX-Umgebung überprüft.

Szenario	Beschreibung
Maschinen Deploy Templates	Der erste Vergleich zwischen einem native ISCSI-Volume und einem ISCSI-SDFS-Volume wird mit dem *deploy* von vier Maschinen aus einem ESX-Template überprüft. Jeweils auf einem ISCSI-SDFS-Volume und einem native ISCSI-Volume werden vier Maschinen nacheinander *deployed*. Nach jedem *deploy* wurde die benötigte Zeit aus dem ESX-Log ermittelt, die zu jeder Aufgabe einen Start-Stempel und Stopp-Stempel führt.
Migration virtueller Maschinen	Bei der zweiten Überprüfung wurde die Geschwindigkeit bei Maschinen-Migrationen gemessen. Vier Maschinen wurden parallel von einem *datastore* auf einen anderen verschoben. Die benötigte Zeit wurde bei einem ISCSI-SDFS-Volume und einem native ISCSI-Volume gemessen.
Kopieren virtueller Maschinen	Im dritten Szenario wurde die Dauer für das Kopieren einer virtuellen Maschine gemessen. In einer ESX-Umgebung wird das Kopieren einer virtuellen Maschine als *cloning* bezeichnet. Die *clones* der virtuellen Maschinen werden oft dazu verwendet, eine reale Testumgebung zu erhalten und dabei die Produktionsmaschinen nicht zu beeinflussen. Diese Aufgabe wurde auf einem ISCSI-SDFS-Volume und native ISCSI-Volume durchgeführt.

4.5.1.1 Maschinen-Deploy Templates

In der ersten Geschwindigkeitsmessung wurde die Zeit gemessen, die ESX für die Erstellung einer virtuellen Maschine aus einem ESX-Template benötigt. Das ESX-Template aus Abschnitt 4.4.3 wurde dafür eingesetzt. Das *deploy* der virtuellen Maschinen erfolgte auf dem ESX-Server CFESX01.

Um das NAS-System (CFSTORE01) zu entlasten, war das ESX-Template auf dem lokalen *datastore* L_CFESX01 von CFEX01 gespeichert. Dadurch wurden die Lesezugriffe auf den lokalen *datastore* und die Schreibzugriffe auf die *datastores* von CFSDF01 und CFSTORE01 je nach Szenario aufgeteilt.

Auf dem ESX-Server CFESX01 wurden zu dem lokalen *datastore* zwei zusätzliche *datastores* konfiguriert. Der *datastore* S_ISCSI_VOL04 vom Storage-System CFSDFS01 und Q_ISCSI_VOL02 vom Storage-System CFSTORE01. Das ISCSI-SDFS-Volume für den *datastore* S_ISCSI_VOL04 wurde zuvor neu, wie in Abschnitt 4.3.3 beschrieben, erstellt.

Bei dieser Performanceanalyse wurde die Zeit zwischen einem Maschinen-Deploy auf S_ISCSI_VOL04 und auf Q_ISCSI_VOL02 gemessen. Nach jedem *deploy* fand eine Zeitmessung über die ESX-Logdateien statt, deren Resultate danach in der Tabelle 11 eingetragen wurden.

Durchlauf	Template -> native ISCSI	Template -> ISCSI-SDFS
1	00:01:46	00:02:12
2	00:01:47	00:01:42
3	00:01:49	00:01:44
4	00:01:46	00:01:40

Tabelle 11: ESX-Template ISCSI vs. SDFS-ISCSI

Der Overhead des DEDUP-Systems wird in Abbildung 28 ersichtlich. Wird die erste virtuelle Maschine auf dem *datastore* S_ISCSI_VOL04 *deployed,* benötigt die SDFS-Appliance mehr Zeit für die Verarbeitung der Daten als für die darauf folgenden. Bei erstem *deploy* einer virtuellen Maschine muss SDFS die Daten in Unikate und Duplikate aufteilen. Bei jeder weiteren Maschineninstanz, die aus dem ESX-Template erstellt wurde, handelte es sich um ein Duplikat.

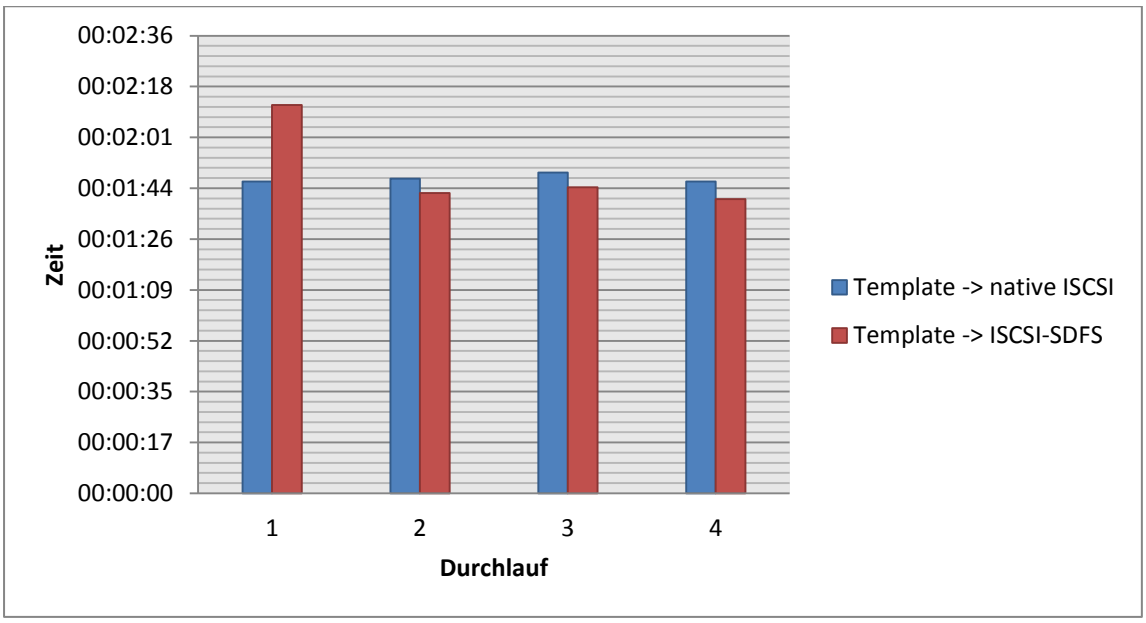


Abbildung 28: Zusammenfassung native ISCSI vs. SDFS-ISCSI

Wird ein ISCSI-SDFS-Volume mit Unikaten konfrontiert, benötigt die SDFS-Appliance mehr Zeit als ein natives ISCSI-Volume. Die Duplikats-Erkennung und das darauf folgende Schreiben der Unikate erfordern ihre Zeit.

Bei jeder weiteren virtuellen Maschine handelt es sich um ein Duplikat. Die SDFS-Appliance erstellte Referenzen, anstatt die Daten auf das NAS-System schreiben zu müssen. Dadurch wurde die SDFS-Variante ab dem ersten Durchlauf schneller. Im Gegensatz dazu bleibt die native ISCSI-Variante vom ersten bis zum letzten Durchlauf bei der benötigten Zeit konstant.

4.5.1.2 Migration virtueller Maschinen

Die zweite Zeitmessung beschäftigte sich mit der Dauer, wenn vier virtuelle Maschinen gleichzeitig migriert werden. Vier Migrationsszenarien wurden durchgeführt:

1. Migration der virtuellen Maschinen von einem native ISCSI-Volume auf ein native ISCSI-Volume.
2. Migration der virtuellen Maschinen von einem native ISCSI-Volume auf ein SDFS-ISCSI-Volume.
3. Migration der virtuellen Maschinen von einem ISCSI-SDFS-Volume auf ein native ISCSI-Volume.
4. Migration der virtuellen Maschinen von einem ISCSI-SDFS-Volume auf ein ISCSI-SDFS-Volume.

Der ESX-Prozess *„migrate datastore“* wurde auf der Maschine CFESX01 durchgeführt. Die vier virtuellen Maschinen von Abschnitt 4.5.1.1 wurden für diese Überprüfung eingesetzt. Die Zeitmessung erfolgte über die ESX-Logdateien.

Für die Überprüfung von ISCSI-SDFS auf ISCSI-SDFS wurde ein neuer *datastore* mit dem Namen S_ISCSI_VOL05 zu dem vorhandenen S_ISCSI_VOL04 angelegt, wie in Abschnitt 4.3.3 beschrieben. Bei der native ISCSI-Überprüfung wurden die *datastores* Q_ISCSI_VOL02 und Q_ISCSI_VOL03 verwendet.

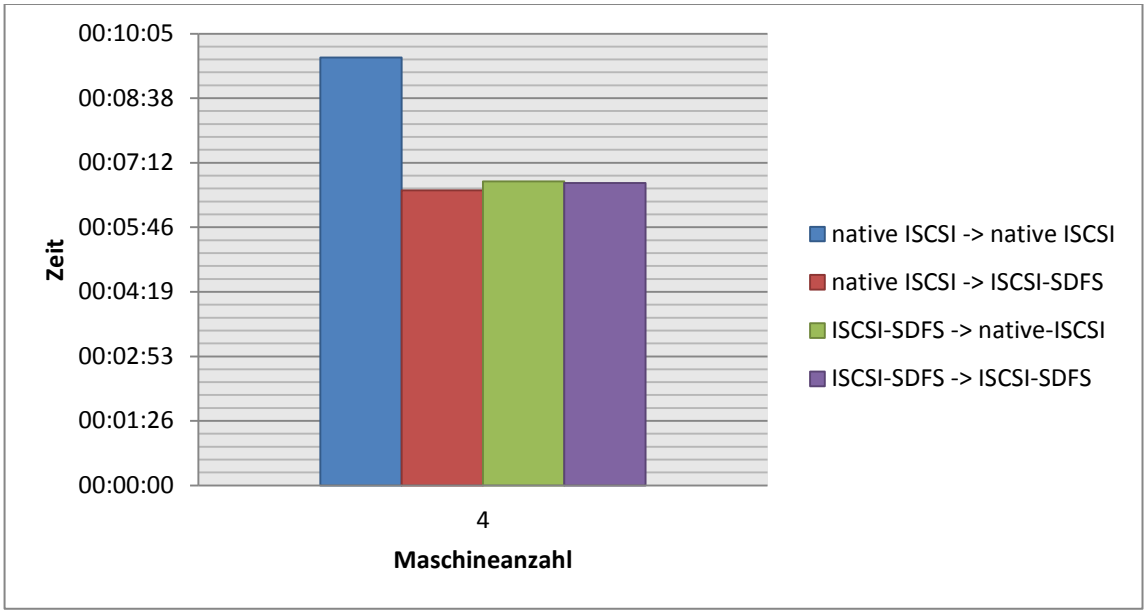


Abbildung 29: Migration virtueller Maschinen native ISCSI vs. ISCSI-SDFS

Bei dieser Überprüfung zeigt sich das NAS-System CFSTORE01 als Flaschenhals, da es die Schreibzugriffe und Lesezugriffe bearbeiten musste. In Abschnitt 4.5.1.1 wurden im Gegensatz die Lesezugriffe vom lokalen *datastore* von CFESX01 bearbeitet.

Das Migrieren einer virtuellen Maschine vom Q_ISCSI_VOL02 auf den Q_ISCSI_VOL03 *datastore* dauerte länger als wenn ein *datastore* von CFSDFS01 eingesetzt wurde (siehe Abbildung 29). Dieses Verhalten wird in Abschnitt 4.5.2 genauer untersucht.

4.5.1.3 Kopieren virtueller Maschinen

In diesem Szenario wurde das Kopieren von virtuellen Maschinen, das sogenannte *cloning,* überprüft. *Clones* werden oft dazu eingesetzt, eine Testumgebung zu erstellen, die zu hundert Prozent der Ausgangsmaschine entspricht. Es können gefahrlos Konfigurationsänderungen oder Programmupdates auf dem *clone* durchgeführt werden, ohne das Original System zu beeinflussen.

Der Prozess des Kopierens ist eine hohe Belastung für das dahinterliegende Storage-System, da durchgehend gleichzeitig gelesen und geschrieben wird. Der *Clone*-Prozess wurde auf dem ESX-Server CFESX01 gestartet. Die *datastores* S_ISCSI_VOL04 als ISCSI-SDFS-Volume und Q_ISCSI_VOL02 als native ISCSI-Volume wurden dafür eingesetzt. In jedem *datastore* wurde als Vorbereitung eine virtuelle Maschine aus dem ESX-Template von Abschnitt 4.4.3 *deployed*. Diese virtuellen Maschinen wurden innerhalb des jeweiligen *datastores* vier Mal kopiert.

In der Tabelle 12 wurden die Zeiten für jeden erstellten *clone* protokolliert. Die Zeiten wurden aus dem ESX-Log entnommen.

Durchlauf	Clone native ISCSI	Clone SDFS-ISCSI
1	00:03:07	00:03:15
2	00:03:04	00:03:15
3	00:03:05	00:03:14
4	00:03:07	00:03:15

Tabelle 12: Kopieren virtueller Maschinen

Wie in Abbildung 30 ersichtlich, war das *clonen* der virtuellen Maschine innerhalb des *datastores* S_ISCSI_VOL04 in allen vier Durchläufen langsamer als im *datastore* Q_ISCSI_VOL02.

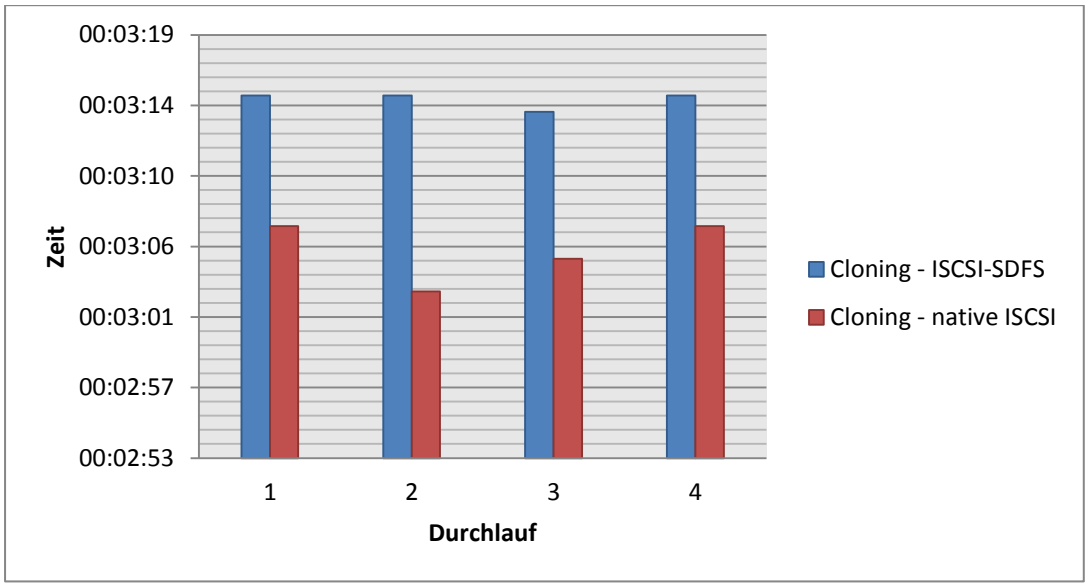


Abbildung 30: Zusammenfassung Kopieren virtueller Maschinen

Das gleichzeitige Lesen und Schreiben innerhalb eines SDFS-Volumes führt in der SDFS-Appliance zu Performance-Einbußen. Auf einem nativen ISCSI-Volume variierten die Zeiten nur leicht, was auf das Caching der ESX-Umgebung und das Storage-System zurückzuführen ist. [BEN2009]

4.5.2 Entlastung Storage-System

In diesem Abschnitt wird untersucht, ob SDFS zur Entlastung eines Storage-Systems eingesetzt werden kann.

Das Beispiel setzte auf ein Szenario, in dem das NAS-System CFSTORE01 stark belastet wurde. Ein ESX-Template wurde nicht auf einem lokalen *datastore* gespeichert, sondern auf dem *datastore* Q_ISCSI_VOL03. Dieses ISCSI-Volume wurde zusätzlich für die zu-

künftigen virtuellen Maschinen verwendet. Eine erhöhte I/O-Last auf CFSTORE01 resultiert daraus.

Es wurden vier virtuelle Maschinen zuerst auf dem *datastore* Q_ISCSI_VOL03 (native ISCSI-Volumes) und danach auf dem *datastore* S_ISCSI_VOL04 (SDFS-ISCSI-Volume) erstellt. Die benötigten Zeiten wurden dem ESX-Log entnommen und in Tabelle 13 eingetragen. Während der virtuellen Maschinenerstellung wurden Screenshots vom grafischen Interface des NAS-Systems und der SDFS-Appliance erstellt.

Durchlauf	Template -> native ISCSI	Template -> ISCSI-SDFS
1	00:02:20	00:02:26
2	00:02:22	00:01:39
3	00:02:24	00:01:39
4	00:02:25	00:01:40

Tabelle 13: I/O SDFS-ISCSI vs. Native ISCSI

Abbildung 31 zeigt die Zusammenfassung der Template-Installation auf den jeweiligen *datastores*. Wie bei der Überprüfung in Abschnitt 4.5.1.1 benötigt ein *datastore* mit einem DEDUP-System im Hintergrund mehr Zeit als ein *datastore* ohne diese Funktion.

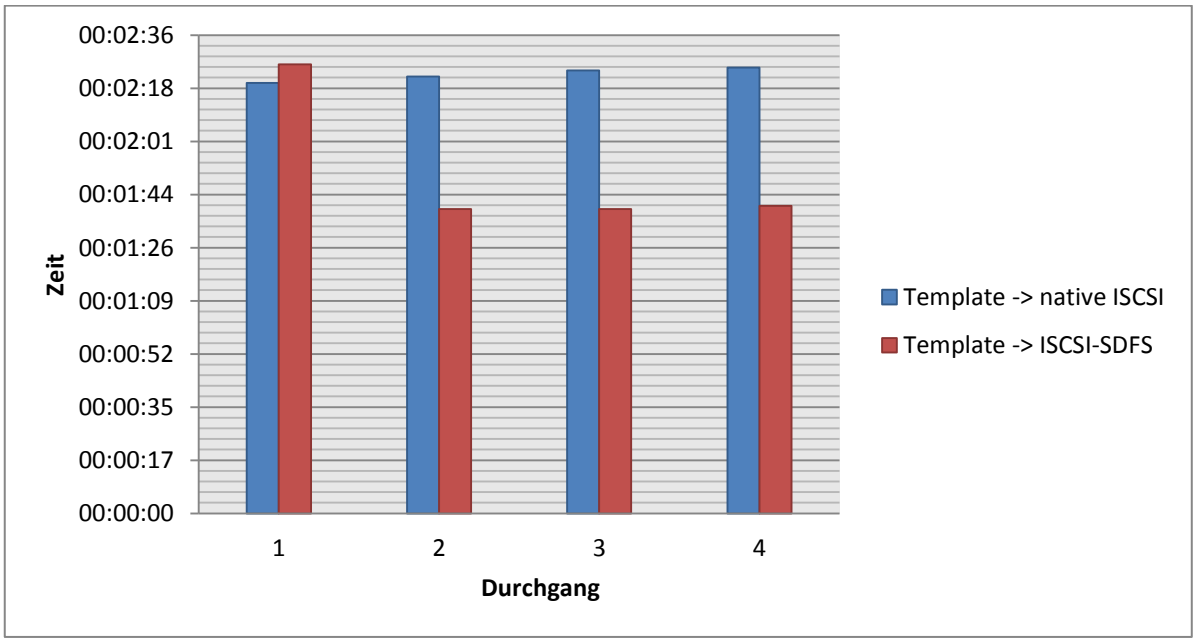


Abbildung 31: I/O Zusammenfassung

Ab dem zweiten Durchgang wird der Unterschied zwischen dem *datastore* Q_ISCSI_VOL03 und dem *datastore* S_ISCSI_VOL04 deutlich. Die zweite virtuelle Maschine musste CFSDFS01 im Gegensatz zum CFSTORE01 nur mehr Referenzen erstellen, da es sich um die gleichen Daten handelte. Dadurch sank die Last auf dem NAS-

System CFSTORE01 deutlich und es standen mehr Ressourcen für andere Aufgaben zur Verfügung.

Abbildung 32 zeigt die Auslastung am Netzwerkadapter des NAS-Systems, bei den ersten vier Durchgängen mit dem *datastore* Q_ISCSI_VOL03. Jedes Rechteck entspricht einer virtuellen Maschineninstallation. Dadurch, dass keine Datendeduplizierung eingesetzt wurde, ist die Auswertung über die vier Durchgänge gleichmäßig. Für das Lesen des ESX-Templates und Schreiben der virtuellen Maschine benötigt ESX mit dem *datastore* Q_ISCSI_VOL03 für jeden Durchgang die gleiche Zeit.

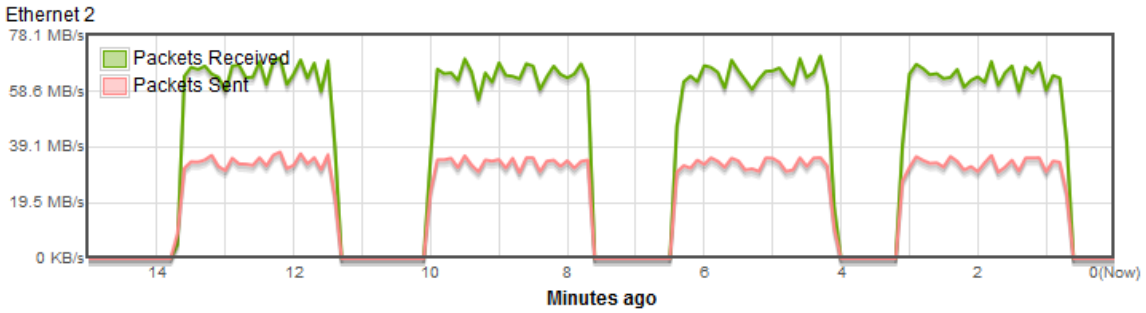


Abbildung 32: Datenrate ESX-Template -> Native ISCSI

In Abbildung 33 wird die Auslastung am Netzwerkadapter vom NAS-System bei der Verwendung des S_ISCSI_VOL03 *datastores* gezeigt. Jedes Rechteck entspricht einer der vier virtuellen Maschinen beim *deploy*. Im ersten Durchgang ist noch ein deutlicher Schreibvorgang (grüne Linie) ersichtlich, der bei den weiteren Durchgängen stark einbricht. Nach dem ersten Durchgang überwiegen die Lesevorgänge (rote Linie) deutlich. Bei den Lesevorgängen handelt es sich wieder um das Auslesen des ESX-Templates.

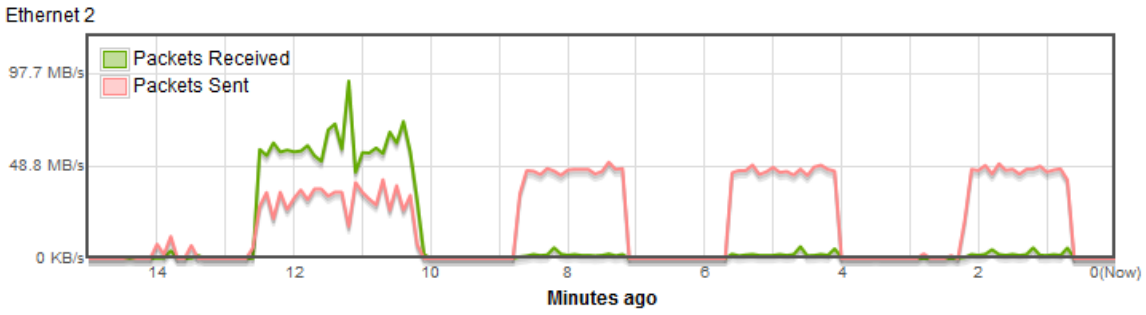


Abbildung 33: Datenrate ESX-Template -> SDFS-ISCSI

In Abbildung 34 und Abbildung 35 wird die aufgezeichnete Deduplizierungsleistung der SDFS-Appliance im ersten und zweiten Durchgang gezeigt. Im ersten Durchgang überwiegt der Anteil der Unikate gegenüber den Duplikaten deutlich.

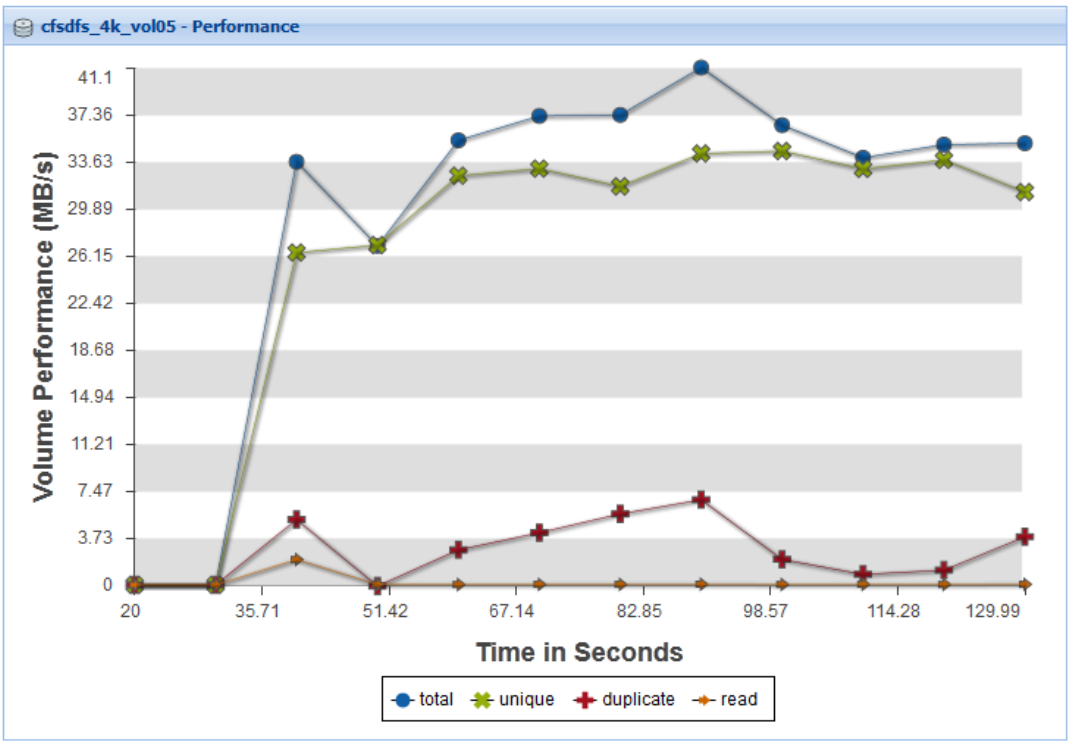


Abbildung 34: SDFS erste virtuelle Maschine

Ab dem zweiten *deploy* der virtuellen Maschine dreht sich das Verhältnis der Unikate und Duplikate um. Fast alle Daten während des *deploys* konnten als Duplikate erkannt werden. Die SDFS-Appliance erstellte Referenzen und konnte die Daten verwerfen.

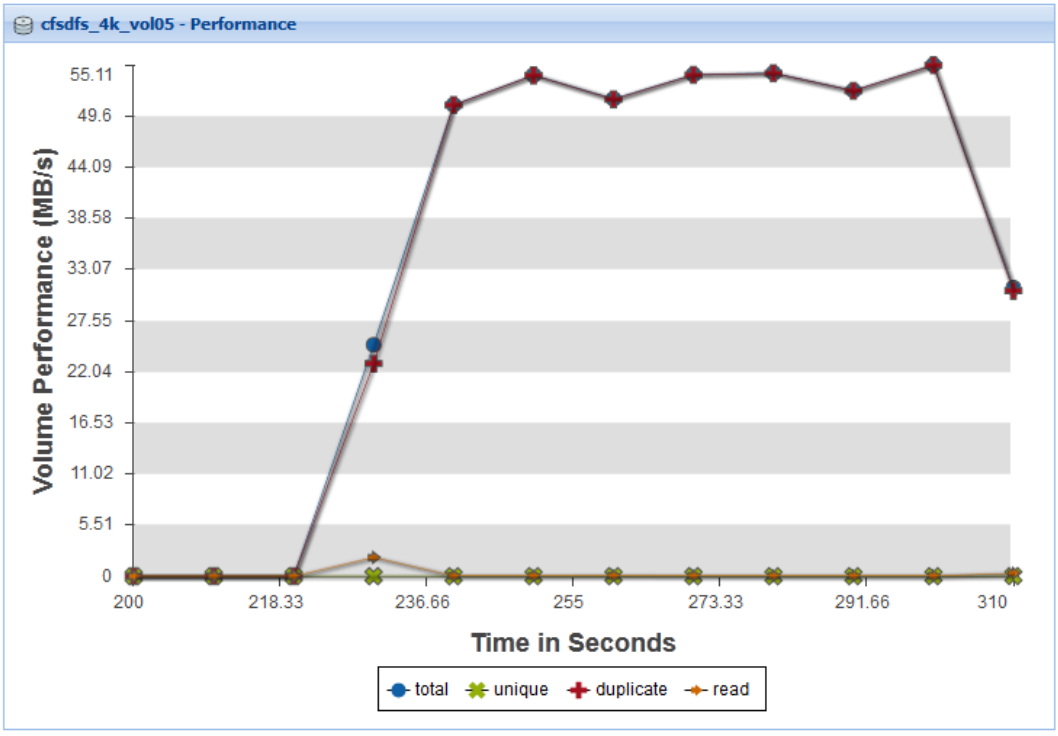


Abbildung 35: SDFS zweite virtuelle Maschine

Ein Storage-System kann durch den Einsatz eines DEDUP-Systems stark entlastet werden. Voraussetzung ist, dass es sich um Daten mit einem hohen Anteil an Duplikaten handelt.

Duplikate werden in einem DEDUP-System durch Referenzen auf die entsprechenden Unikate ersetzt und anschließend verworfen. Dieser Prozess führt zu dem verminderten Schreibprozess am Storage-System.

4.5.3 SDFS-Performance Metadaten

Für die Verwaltung der Unikate und Duplikate benötigt SDFS zusätzlichen Speicherplatz. Dieser Speicher wird genutzt, um Informationen zu den gespeicherten Daten, Hashwerten und Referenzen abzulegen. Diese Informationen werden unter dem Begriff Metadaten[32] zusammengefasst.

Metadaten wachsen je nach Auslastung eines SDFS-Volumes. Je mehr Daten abgelegt werden, umso größer wird der benötigte Speicherplatz für die Metadaten. Dadurch wird auch die Suche nach Duplikaten und Unikaten aufwendiger, da es immer mehr Hashwerte zu vergleichen gilt.

In diesem Abschnitt wird überprüft, wie sich das Wachstum der Metadaten auf die Performance der SDFS-Appliance auswirkt. Die Performance wird in zwei Szenarien untersucht, einmal mit zufälligen Daten, die sich schlecht deduplizieren lassen, und einmal mit Daten, die aus Nullen bestehen. Diese lassen sich durch die immer wiederkehrende Null sehr gut deduplizieren.

Für die Überprüfung wurde ein ESX-Template kreiert, das bis auf die Größe der Festplatte und der Partitionierung dem ESX-Template aus Abschnitt 4.4.3 entspricht. Anstatt 150 GB Festplattenspeicher wurde der Speicher auf 230 GB erweitert, um die SDFS-Appliance auszulasten. Die Gesamtkapazität von 250 GB war nicht möglich, da SDFS und ESX zusätzlichen Speicherplatz für die Verwaltung benötigen.

Die Konfiguration der virtuellen Maschine ist in Anlage 5 ersichtlich. Unter Anlage 6 ist die Konfiguration des verwendeten ISCSI-SDFS-Volumes gelistet. Bis auf die Größe von 250 GB für den Unikats- und Duplikatsspeicher ist das Volume ident mit der Konfiguration aus Anlage 2.

Mit dem ESX-Template wurden zwei virtuelle Maschinen mit den Namen CFVIRT25 und CFVIRT26 erstellt. CFVIRT26 nutzte den *datastore* Q_ISCSI_VOL03, der vom NAS-System (CFSTORE01) bereitgestellt wurde und damit ein native ISCSI-Volume. CFVIRT25 wurde auf dem *datastore* S_ISCSI_VOL09 vom SDFS-System (CFSDFS01) abgelegt (SDFS-ISCSI-Volume).

[32] Bei Metadaten handelt es sich um Daten, die andere Daten beschreiben (siehe http://de.wikipedia.org/wiki/Metadaten).

Abbildung 36 zeigt den Aufbau für die Überprüfung des ersten Szenarios. Der ESX-Server CFESX01 stellt über seinen lokalen *datastore* (L_CFESX01) die Daten mit dem Inhalt aus Nullen zur Verfügung. CFESX03 wiederum stellt mit dem lokalen *datastore* (L_CFESX03) die zufälligen Daten bereit. Jeder lokale *datastore* wurde mit zwanzig Zehn-Gigabyte-Dateien beschrieben. Die Dateigröße wurde gewählt, um das Caching über den Hauptspeicher der virtuellen Maschinen zu verhindern.

Für die Generierung der Daten wurden die Linux-Geräte */dev/null* für die Nullen und das Linux-Gerät */dev/urandom* für die zufälligen Daten verwendet. Um das NAS-System CFSTORE01 zu entlasten, wurden die Dateien auf den jeweiligen lokalen *datastore*s der ESX-Server abgelegt. In der Anlage 7 ist das Script für die Erstellung der Dateien gelistet. Für die Zeitmessung kam das Linux-Programm *time*[33] zum Einsatz. Um die Beeinflussung des Hauptspeichers zu verkleinern, wurden vor jedem Kopiervorgang die Speicher der virtuellen Maschine entleert.

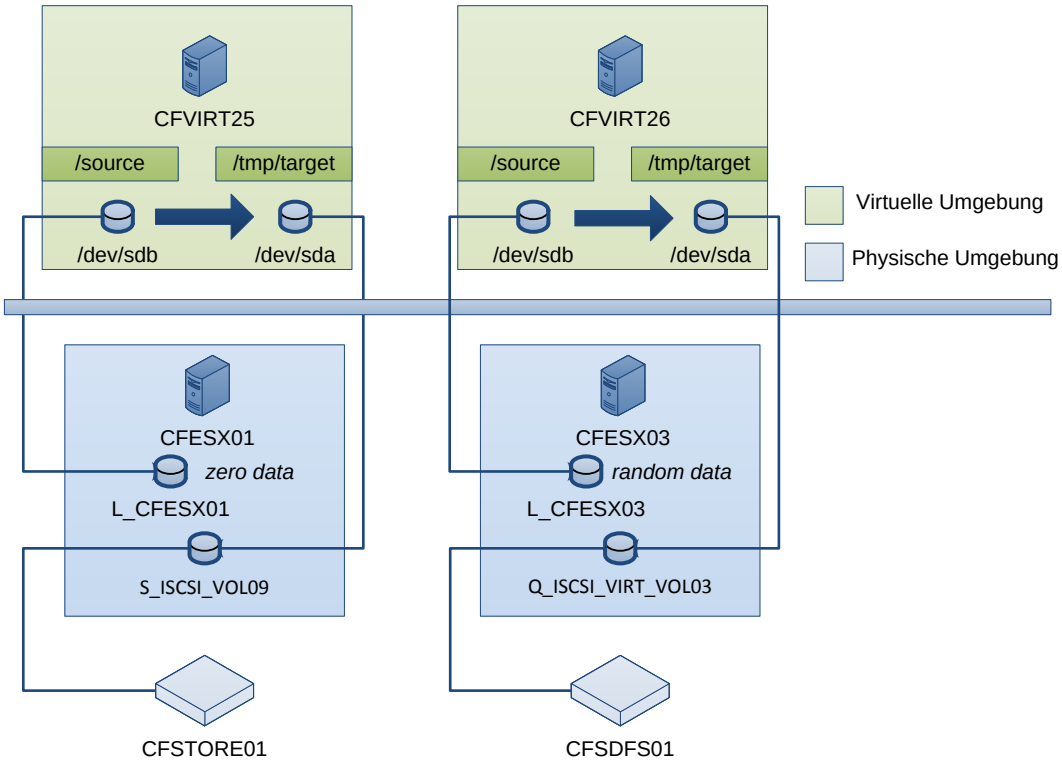


Abbildung 36: Metadaten Testaufbau

Die virtuellen Maschinen nutzten für die Betriebssysteme die *datastores* S_ISCSI_VOL09 und Q_ISCSI_VIRT_VOL03. L_CFESX01 und L_CFESX03 sind die lokalen *datastores* der ESX-Server und wurden als zweite Festplatte in die virtuellen Maschinen eingebunden. Mit dem Script aus Anlage 7 wurden die Dateien innerhalb der virtuellen Maschine vom lokalen *datastore* (/dev/sdb) der jeweiligen ESX-Server auf die *datastores* S_ISCSI_VOL09 und Q_ISCSI_VIRT_VOL03 (dev/sda) geschoben. Bei jedem Kopiervor-

[33] Bei *time* handelt es sich um ein einfaches Linux-Programm das die Zeit für die Ausführung eines Befehls misst.

gang wurde die Zeit gemessen, um die benötigten Zeiten von der ersten bis zur zwanzigsten Kopie zu erhalten.

Im ersten Szenario wurden innerhalb der Maschine CFVIRT25 die *zero data* vom *datastore* L_CFESX01 kopiert. In der Maschine CFVIRT26 wurden *die random data* vom datastore L_CFESX03 kopiert.

Nach der Überprüfung und Protokollierung der Zeiten wurden beide Maschinen und das SDFS-Volume S_ISCSI_VOL09 gelöscht. Nach dem Löschen wurden die virtuellen Maschinen aus dem ESX-Template und das SDFS-Volume S_ISCSI_VOL09 neu erstellt. Die virtuellen Maschinen wurden umgekehrt auf die ESX-Server verteilt. Dadurch konnte das zweite Szenario überprüft werden. CFVIRT25 kopierte die *random data* und CFVIRT26 *die zero data*.

In Tabelle 14 wird eine Übersicht der zwei Testszenarien gegeben.

Szenario	Virtuelle Maschine	Datastore OS	Storage-System	Datastore Source	Data-Type	ESX-Server
1	CFVIRT25	S_ISCSI_VOL09	CFSDFS01	L_CFESX03	RANDOM	CFESX03
	CFVIRT26	Q_ISCSI_VIRT_VOL03	CFSTORE01	L_CFESX01	ZERO	CFESX01
2	CFVIRT26	Q_ISCSI_VIRT_VOL03	CFSTORE01	L_CFESX03	RANDOM	CFESX03
	CFVIRT25	S_ISCSI_VOL09	CFSDFS01	L_CFESX01	ZERO	CFESX01

Tabelle 14: Metadaten Szenarien

In Abbildung 37 werden die vier Ergebnisse der zwei Szenarien zusammenfassend dargestellt.

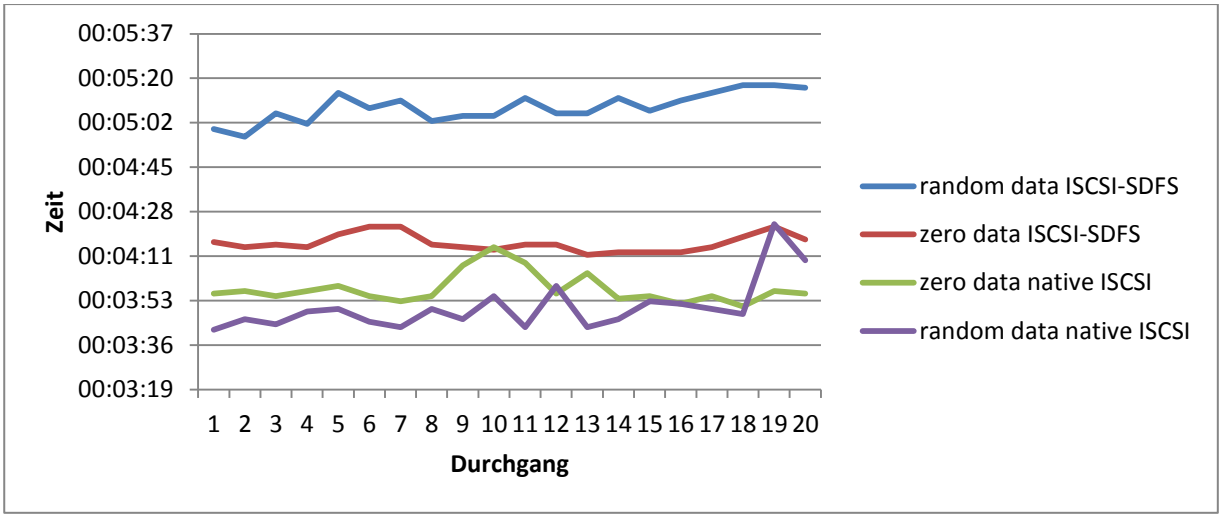


Abbildung 37: Metadaten Zusammenfassung

CFVIRT25 nutzte als *datastore* ein native ISCSI-Volume und damit keine Datendeduplizierung. In beiden Szenarien (*random data/zero data*) war die benötigte Zeit für das Kopieren der Daten konstant. Im Gegensatz dazu steht die virtuelle Maschine CFVIRT26. Diese nutzte einen *datastore* vom Typ SDFS-ISCSI und damit eine Datendeduplizierung. Im Szenario, in dem die Daten aus Nullen bestanden, stieg die benötigte Zeit nur leicht an. Sie ist vergleichbar mit den Zeiten vom nativen ISCSI-Volume.

Die gemessenen Zeiten der virtuellen Maschine CFVIRT26 veränderten sich stark im Szenario mit den zufälligen Daten. SDFS konnte bei dieser Art von Daten keine hohe DEDUP-Ratio erreichen. Die Daten wurden Großteils als Unikate gespeichert. Dieser Prozess benötigte mehr Zeit als die anderen Varianten. Eine signifikante Beeinflussung durch die Metadaten trat nicht auf.

Zusammenfassend beeinflusst die Verwaltung der Unikate und Duplikate die Performance nicht maßgeblich, auch wenn der Unikatsspeicher ausgereizt wird. Die benötigte Zeit für die Speicherung eines Unikats ist deutlich höher als der Verwaltungsaufwand der durch einen vollen Unikatsspeicher ausgelöst wird.

4.6 SDFS-Konfiguration

Die Installation der SDFS-Appliance wurde mit den Bordmitteln von ESX durchgeführt. Nach der Installation war die Grundkonfiguration über die SDFS-Management-Konsole möglich. Diese umfasst die Konfiguration der Netzwerkanbindung, das Setzen des Administrationspassworts und die Anbindung an die ESX-Umgebung.

Das Erstellen eines SDFS-Volumes bis zur Anbindung über ISCSI oder NFS an die ESX-Umgebung wurde grafisch durch die SDFS-Management-Konsole unterstützt. Die SDFS-Volumes konnten über mehrere Parameter an die jeweilige Umgebung angepasst werden. Die Parameter umfassten die Definition der Unikats- und Duplikatsspeicher bis zu einem Passwortschutz der SDFS-Volumes.

Bis zu diesem Punkt wurden keine Linux-Kenntnisse oder spezielle Kommandozeilen-Befehle benötigt. Alle ausgeführten Einstellungen waren per Maus und den entsprechenden Eingabemasken möglich.

Einschränkungen ergaben sich bei weiterführenden Konfigurationen. Die Ansicht von Logdateien über die SDFS-Management-Konsole war nicht möglich. Diese konnten nur über die Kommandozeile angezeigt werden. Einstellungen der Zeitzone waren ebenfalls nur über die Kommandozeile möglich.

Eine weitere Einschränkung betraf die Kapazität des Unikatsspeichers. Dieser war bei der SDFS-Appliance auf 250 GB beschränkt. Nimmt man eine 10:1 DEDUP-Ratio an, können 2,5 TB an Daten gespeichert werden. Die Einschränkung lässt sich über weitere SDFS-Appliances, die parallel arbeiten, beheben. Durch den direkten Eingriff in die SDFS-Appliance, ist es zusätzlich Möglich den Unikatsspeicher anzuheben. Dafür werden Linux-Kenntnisse vorausgesetzt.

Innerhalb der SDFS-Appliance sind die Konfigurationsdateien der SDFS-Volumes im Verzeichnis */etc/sdfs* abgelegt. Die Logdateien werden im Verzeichnis */var/log/sdfs* gespeichert. Für jedes SDFS-Volume wird eine separate Logdatei geführt. Die Verwaltung des SDFS-Volumes und die Unikate sind unter */opt/sdfs* abgelegt. Jedes SDFS-Volume erhält ein eigenständiges Verzeichnis. Die Verzeichnisstruktur, Logdateien und die verwendeten Konfigurationsdateien, die für die Testumgebung eingesetzt wurden, sind in Anlage 1 ersichtlich.

Im Betrieb ist es möglich, die Größe des Duplikatsspeichers anzupassen. Eine Änderung der Größe des Unikatsspeichers ist online wie offline nicht möglich. Erstellte ISCSI-Targets lassen sich im Betrieb vergrößern.

4.7 SDFS-Stabilität

In diesem Abschnitt wird die Stabilität der SDFS-Appliance bewertet. Die Bewertung stützt sich auf die gesammelten Erfahrungen aus den Abschnitten 4.3, 4.4 und 4.5. Es wird zwischen den zwei Anbindungsmöglichkeiten ISCSI und NFS unterschieden, da unterschiedliche Fehlerbilder existierten.

4.7.1 SDFS NFS-Anbindung

In den ersten Überprüfungen der SDFS-Appliance konnten keine Instabilitäten entdeckt werden. Es wurden einzelne virtuelle Maschinen nacheinander auf ein SDFS-Volume geschrieben, um die Datenreduktion zu ermitteln. Bei der Erhöhung der I/O-Last durch paralleles Schreiben und Lesen mehrerer virtuellen Maschinen auf einem SDFS-Volume traten in der Logdatei folgende Info-Meldungen auf, siehe Abbildung 38.

```
[Thread-426] INFO sdfs  - trying to read again
[Thread-406] INFO sdfs  - trying to read again
[Thread-487] INFO sdfs  - trying to read again
```

Abbildung 38: Fehlermeldung read again

Diese Meldung führte nicht sofort zu einer Beeinträchtigung der Funktion. Wenn die Last über mehrere Minuten konstant hoch gehalten wurde, häuften sich diese Meldungen und der Zugriff der ESX-Umgebung auf das SDFS-Volume war nicht mehr möglich. Die Bereitstellung des SDFS-Volumes über das NFS-Protokoll funktionierte nicht mehr. Dadurch schaltete die ESX-Umgebung das betroffenen *datastore* offline.

Damit das SDFS-Volume wieder funktionierte, musste über die SDFS-Management-Konsole das defekte SDFS-Volume ausgehängt und wieder eingehängt werden. Danach führte die SDFS-Appliance eine Filesystemüberprüfung durch und das SDFS-Volume war danach für die ESX-Umgebung wieder erreichbar. Datenverlust konnte in der Praxisumgebung keiner nachgewiesen werden. Überprüft wurden die virtuellen Maschinen auf Da-

tenverlust mit dem UNIX-Befehl *rpm –Va*[34]. Dieser Befehl kontrolliert die installierten Programmpakete über einen Hashwert der bei einer RPM-Installation in einer Datenbank gespeichert wird.

Dieses Problem konnte auch mit tiefer gehenden Recherchen nicht gelöst werden.

4.7.2 SDFS ISCSI-Anbindung

Um weitere Überprüfungen durchführen zu können, wurde versucht, die NFS-Anbindung durch die ISCSI-Anbindung zu ersetzen.

Diese Verbindungsart führte zu sofortigem I/O-Fehler innerhalb der ESX-Umgebung, obwohl über die SDFS-Management-Konsole keine Fehler erkannt wurden. Die Überprüfung der Log-Dateien vom SDFS-Volume gab keinen Aufschluss über dieses Verhalten. Eine weitere Überprüfung der Logdateien vom eingesetzten Betriebssystem der SDFS-Appliance ergab folgende Meldungen, die in Abbildung 39 ersichtlich sind.

```
sdfsnas kernel: [438598.751712] WRITE_SAME w/o UNMAP bit not supported for Block Discard Emulation
sdfsnas kernel: [438623.167876] WRITE_SAME w/o UNMAP bit not supported for Block Discard Emulation
sdfsnas kernel: [438922.293931] WRITE_SAME w/o UNMAP bit not supported for Block Discard Emulation
sdfsnas kernel: [439221.428941] WRITE_SAME w/o UNMAP bit not supported for Block Discard Emulation
```

Abbildung 39: Fehlermeldung WRITE_SAME

Bei WRITE_SAME handelt es sich um ein spezielles SCSI-Command, das ESX einsetzt, um *datastores* mit Nullen (zeroing) zu beschreiben, siehe [VMB2012]. Anstatt einzelne Datenblöcke mit Nullen an das Storage-System zu schicken, werden ein Startblock und Endblock definiert und einmalig an das Storage-System geschickt. Das Storage-System beschreibt selbstständig die Blöcke zwischen Startblock und Endblock mit den definierten Daten und schickt danach den Status zurück an die ESX-Umgebung. Der I/O wird zwischen dem Storage-System und der ESX-Umgebung deutlich verringert. Diese Funktion wird bei ESX ab der Version 5 eingesetzt und wird abgeschaltet, wenn das Storage-System das SCSI-Command WRITE_SAME nicht unterstützt.

Die ISCSI-Implementierung in der SDFS-Appliance kann diesen SCSI-Command in der derzeitigen Version nicht umsetzen. Ein Fehler im Linux-Kernel führt zu einem falschen *return code* an die ESX-Umgebung. Dieser falsche *return code* verhindert die Abschaltung des Features in der ESX-Umgebung und führt zu den I/O-Fehlern. In der Linux-Kernel *mailing list* befindet sich die Fehlerbeschreibung. [KERN2012]

Um die SDFS-Appliance in der aktuellen Version mit ISCSI in einer ESX-Umgebung ab der Version 5 zu betreiben, muss das Feature deaktiviert werden. Der SCSI-Command WRITE_SAME kann über den *vStorage APIs for Array Integration* (VAAI) in der ESX-Umgebung aus- und eingeschaltet werden. In der Literaturquelle [VMW2012] werden die

[34] Bei RPM handelt es sich um einen weit verbreiteten Paketmanager für Linux-Umgebungen (siehe http://www.rpm.org/).

benötigten Schritte dokumentiert, um VAAI zu deaktivieren. Nach der Deaktivierung arbeitete die SDFS-Appliance mit der ESX-Umgebung zusammen.

4.7.3 Stabilität allgemein

Die SDFS-Appliance arbeitete bis auf die anfänglichen Stabilitätsprobleme mit ISCSI und NFS unauffällig. Während der Überprüfungen wurden bis zu sechs SDFS-Volumes angelegt und diese immer wieder gelöscht und neu erstellt. Maximal drei SDFS-Volumes konnte parallel betrieben werden. Durch die physikalische Einschränkung von acht Gigabyte sind mehr SDFS-Volumes im parallelen Betrieb nicht möglich gewesen.

Die SDFS-Management-Konsole reagierte immer zufriedenstellend auf die Konfigurationsänderungen. Bewusste Falschkonfigurationen, wie den Namen eines vorhandenen SDFS-Volumes zu verwenden, wurden erkannt und mit einer Fehlermeldung blockiert.

Datenverluste konnten in den Testläufen keine festgestellt werden. Geprüft wurden die ESX-Logdateien auf I/O-Fehler und die virtuellen Maschinen selbst mit dem Befehl Linux Befehl *rpm –VA*.

4.7.4 SDFS Konsistenz-Check

Die Instabilitäten, die bei NFS und ISCSI auftraten, führten dazu, dass SDFS-Volumes nicht geordnet herunterfahren konnten. Wird ein SDFS-Volume in diesem Zustand wieder eingehängt, erkennt SDFS den Status und führt einen Konsistenz-Check durch. Die Dauer der Überprüfung hängt von der Größe des Unikatsspeichers ab. Je größer der Speicher, umso länger dauerte diese Überprüfung.

```
[main] INFO sdfs  - DSE did not close gracefully, running consistancy check
[main] WARN sdfs  - Running Consistancy Check on DSE, this may take a while
[main] WARN sdfs  - Succesfully Ran Consistance Check for [8412632] records, recovered [0]
```

Abbildung 40: SDFS Konsistenz-Check

In Abbildung 40 wird der Konsistenz-Check in der Laborumgebung gezeigt. Der Unikatsspeicher hatte eine Größe von 33432 MB und die Überprüfung dauerte sechs Minuten. Berechnet man die Durchsatzgeschwindigkeit, erreichte die Überprüfung 92,9 MB/sec, was in etwa die Geschwindigkeit eines Gigabit-Netzwerks widerspiegelt. Diese Dauer der Überprüfung kann in einer produktiven Umgebung zu Problemen führen, wenn ein Ausfall einer IT-Dienstleistung nur für eine bestimmte Zeit toleriert werden kann. Eine Schnellüberprüfung eines SDFS-Volumes wird nicht angeboten.

4.8 SDFS-Installation

Mark Silverberg ermöglicht es, SDFS in verschiedenen Varianten zu testen. Es besteht die Auswahl zwischen dem Quellcode, diversen Binärpaketen oder einer SDFS-Appliance.

Für die Diplomarbeit wurde die SDFS-Appliance aus zwei Gründen gewählt. In der überprüften Version ist die SDFS-Management-Konsole nur in der SDFS-Appliance verfügbar und die Konsole bietet Schnittstellen zwischen ESX und SDFS an.

Die Installation der SDFS-Appliance konnte in wenigen Schritten mit der Funktion „*Deploy OVF Template*“ vom ESX VSphere-Client durchgeführt werden. Konfigurationsänderungen nach der Installation wie die Netzwerkkonfiguration und das setzen eines Administrationspasswort sind über die SDFS-Management-Konsole möglich.

Tiefer gehende Einstellungen müssen direkt am Linux-System durchgeführt werden.

4.9 SDFS-Support

Für SDFS wird aktuell kein kommerzieller Support angeboten.

Auf freiwilliger Basis beantworten der Entwickler und die Community Fragen und geben ihre Erfahrungsberichte wider. Anlaufstellen für Fragen, Fehlermeldungen oder Wünsche ist die Projektseite (http://www.opendedup.com). Auf dieser Seite existieren weiterführende Links zu den Diskussionsgruppen und der Fehlerliste.

Fehler und Wünsche können unter der Seite
http://code.google.com/p/opendedup/issues/list gemeldet werden.

Anlaufstelle für Fragen ist die öffentliche Diskussionsgruppe unter der Seite:
https://groups.google.com/forum/?fromgroups#!forum/dedupfilesystem-sdfs-user-discuss

5 Ergebnisse und Ausblick

Ergebnisse der Diplomarbeit und die Bewertung des Autors werden in diesem Kapitel zusammengefasst. Ein Ausblick auf zukünftige Entwicklungen und deren Möglichkeiten der Software SDFS wird gegeben.

5.1 Ergebnisse

Der Einsatz eines kommerziellen DEDUP-Systems im Bereich der Datensicherung weckte das Interesse des Autors an dieser Form der Datenreduzierung. Der Einsatz der Virtualisierungsumgebung ESX der Firma VMware beim Kunden und das ständige Wachstum der Anzahl von virtuellen Maschinen führten zu dem Thema Datendeduplizierung in einer Virtualisierungsumgebung. Um das Potenzial dieser Technologie abschätzen zu können, wurden Möglichkeiten gesucht, dieses näher zu betrachten.

Grundlage für die Diplomarbeit war die Recherche, ob es technisch realisierbar ist, mit freier Software dieses Ziel zu erreichen. Die Softwareprojekte LessFS und SDFS ermöglichten ein weiteres Vorgehen sowie eine Definition des Diplomarbeitsthemas.

Um das geeignete Softwareprojekt auszuwählen, wurden die Projekte nach definierten Auswahlkriterien verglichen und bewertet. Nach Abschluss der Bewertung stellte sich das Software-Projekt SDFS als geeigneter Kandidat heraus. Hauptgründe für diese Entscheidung waren die enge Verzahnung von SDFS mit dem Virtualisierungsprodukt ESX, die gute Dokumentation und das Angebot einer Software-Appliance.

Nach der Festlegung des Softwareprojekts wurde ein Anforderungskatalog erstellt, um die Software als geeignet oder nicht geeignet einzustufen. Für die Überprüfung der Software SDFS wurde eine Laborumgebung aufgebaut, die von der Funktion derjenigen der produktiven Umgebung beim Kunden entsprach. Die Umgebung diente dazu, die Software SDFS unabhängig und gefahrlos für die produktive Umgebung zu testen.

Die Überprüfung selbst erfolgte in verschiedenen Szenarien, die die typischen Aufgaben einer ESX-Umgebung widerspiegeln. Diese Szenarien wurden hinsichtlich der DEDUP-Ratio und Performance untersucht.

Während der Untersuchung der ersten Szenarien stellten sich schwere Mängel bei der Anbindung der SDFS-Volumes mit den Netzwerkprotokollen NFS und ISCSI heraus. Für ein weiteres Vorgehen musste eine Lösung für diese Mängel gefunden werden. Eine Stabilisierung der NFS-Anbindung mit den vorgegebenen Mitteln der SDFS-Appliance war nicht möglich. Die ISCSI-Anbindung konnte nach Konfigurationsänderungen an der ESX-Umgebung für die weiteren Versuche eingesetzt werden.

Es folgte die Überprüfung der Software anhand der ausgewählten Testszenarien.

5.2 Bewertung der Arbeit

Nach Beendigung dieser Arbeit war das Softwareprojekt SDFS erfolgreich in einer ESX-Umgebung integriert. Um einen stabilen Betrieb zu erhalten, mussten mehrere Punkte beachtet werden. Der Einsatz des Protokolls NFS wurde durch massive Stabilitätsprobleme ausgeschlossen. ISCSI hingegen erforderte durch einen Fehler im Betriebssystemkern der SDFS-Appliance Konfigurationsänderungen an der ESX-Umgebung. Ein falscher *return code* der SDFS-Appliance verursachte ein falsches Verhalten der ESX-Umgebung und endete in Schreib/Lese-Fehlern.

Werden diese Punkte beachtet, ist die Einbindung ohne weitere Hürden möglich. Der angebotene Storage der SDFS-Appliance verhält sich im Betrieb aus der Sicht der ESX-Umgebung wie ein Standard-Storage-System. Die Datendeduplizierung läuft transparent im Hintergrund. In dieser Umgebung war es möglich, SDFS in allen definierten Szenarien in der Laborumgebung einzusetzen und zu überprüfen.

Bei der DEDUP-Ratio zeigte SDFS Schwächen bei manuellen Maschineninstallationen. Bei vier virtuellen Maschinen mit identer Softwareausstattung erreichte SDFS eine DEDUP-Ratio von 2,4:1. Vernachlässigt man die Verwaltung, war in diesem Beispiel eine DEDUP-Ratio von 4:1 möglich. Dieser Wert konnte mit einer ESX-Template-Maschineninstallation annähernd erreicht werden. Nach vier installierten virtuellen Maschinen aus dem ESX-Template betrug die DEDUP-Ratio 3,8:1.

Wird die Performance zwischen einem Standard-Storage-System und der SDFS-Appliance verglichen, fallen die Ergebnisse sehr unterschiedlich aus. Ein neu angelegtes SDFS-Volume, das mit einer virtuellen Maschine beschrieben wird, ist um ca. 24% langsamer als ein Standard-ISCSI-Volume. Wird der Vorgang mit der gleichen virtuellen Maschine wiederholt, dreht sich das Ergebnis. Der Vorgang beim SDFS-Volume ist danach um ca. 5% schneller. Vorausgesetzt, der SDFS-Appliance stehen genügend Ressourcen für den DEDUP-Prozess zur Verfügung und das Storage-System ist der limitierende Faktor.

Diese gegenteilige Messung entsteht durch den Ablauf einer Datendeduplizierung. Der erste Durchgang setzt sich aus dem DEDUP-Prozess und dem Schreiben der Unikate zusammen. Beim zweiten Durchgang entfällt das Schreiben der Unikate, da es sich um die gleichen Daten handelt. Anstatt die Daten zu speichern, wurden Referenzen auf die Unikate erstellt. Dieser Prozess führte zur Entlastung des Storage-Systems und einer schnelleren Abarbeitung der Aufgabe.

5.3 Ausblick

Für eine Produktivumgebung ist die SDFS-Appliance in der überprüften Version noch nicht einsatzfähig. Hauptkritikpunkte sind die Instabilitäten mit dem Protokoll NFS und die notwendigen Änderungen an der ESX-Konfiguration, um ISCSI einsetzen zu können.

Ein weiterer Kritikpunkt betrifft die maximale Speicherkapazität für den Unikatsspeicher von 250 GB, der ohne Linux-Kenntnisse nicht erweiterbar ist. Nimmt man eine DEDUP-Ratio von 10:1 an, kann eine SDFS-Appliance maximal 2,5 TB an Daten halten. Wenn mehr Datenspeicher gefordert wird, müssen weitere SDFS-Appliances eingesetzt werden.

Positiv hebt sich die webbasierende SDFS-Management-Konsole hervor. Die Oberfläche ist klar strukturiert und bietet alle Funktionen, um einen SDFS-Storage in eine ESX-Umgebung zu integrieren. Fehlererkennung bei falscher Eingabe und ein Diagramm der aktuellen Datenrate runden die Oberfläche ab.

Für den produktiven Einsatz fehlen aber wichtige Punkte im Bereich der Überwachung und Administrierbarkeit.

- Eine Ansicht der Log-Daten vom Betriebssystem oder der SDFS-Volumes ist nicht integriert. Diese können bis jetzt nur über die Kommandozeile betrachtet werden.
- Notifikationen bei einem Fehlerfall sind nicht möglich. Wie zum Beispiel eine E-Mail-Benachrichtigung.
- Ein Update oder Upgrade der SDFS-Appliance ist nicht implementiert. Updates müssen manuell durchgeführt werden. Eine offizielle Dokumentation zum Stand der Arbeit existiert nicht.
- Die Überprüfung eines SDFS-Volumes wird über den kompletten Unikatsspeicher durchgeführt. Diese Überprüfung benötigt Zeit und Ressourcen. Eine schnellere Überprüfung eines SDFS-Volumes ist nicht möglich.
- Der Unikatsspeicher eines SDFS-Volumes ist nach der Erstellung nicht erweiterbar. Wird die geschätzte DEDUP-Ratio unterschritten, kann diese Fehleinschätzung nur durch ein Neuanlegen oder mit einem weiteren SDFS-Volume korrigiert werden.

Die Punkte können von einem Experten im Linux-Bereich gelöst werden, sind aber für einen Anwender ein sehr schwieriges Unterfangen.

Diese Diplomarbeit zeigt die Möglichkeiten und Potenziale eines DEDUP-Systems auf Basis freier Software in einer Virtualisierungsumgebung. SDFS ist sehr nützlich, um in einer Testumgebung Erfahrungen mit der Datenreduktion, Geschwindigkeit und dem Verhalten eines DEDUP-Systems zu sammeln. Die gute Dokumentation und das Angebot der Software-Appliance erleichtern den Einstieg.

Eine Empfehlung, SDFS in einer Produktivumgebung einzusetzen, kann durch die Mängel und die fehlenden Optionen in der Überwachung und Administrierbarkeit nicht gegeben werden.

Literatur

[BENC2009] Cisco: A Nine Year Study of File System and Storage. URL: http://www.fsl.cs.sunysb.edu/docs/fsbench/fsbench-tr.pdf, verfügbar am 15.10.2012

[CIS2011] Cisco: Cisco WAAS-Software Technical Overview. URL: http://www.cisco.com/en/US/prod/collateral/contnetw/ps5680/ps6870/prod_white_paper0900aecd8051d5b2.pdf, verfügbar am 16.08.2012

[COL2010] eXdupe: Risk of hash collisions in data deduplication. URL: http://www.exdupe.com/collision.pdf, verfügbar am 16.08.2012

[DATA2009] Loch, Semerci: Data Deduplication im Backup/Recovery Umfeld. URL: http://winfwiki.wifom.de/index.php/Data_Deduplication_im_Backup/Recovery_Umfeld, verfügbar am 16.08.2012

[KIT2008] Swoboda, Joachim; Spitz, Stephan; Pramateftakis, Michael: Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen – eine Einführung, Vieweg+Teubner Verlag 2008, ISBN 3-83480-248-4

[IBMC2012] IBM: IBM Tivoli Storage Manager für Unix and Linux Clients für Sicherung/Archivieren. URL: http://pic.dhe.ibm.com/infocenter/tsminfo/v6r3/topic/com.ibm.itsm.client.doc/b_ba_guide_unx_lnx.pdf, verfügbar am 16.08.2012

[IBMS2012] IBM: IBM Tivoli Storage Manager for Linux. URL: http://pic.dhe.ibm.com/infocenter/tsminfo/v6r3/topic/com.ibm.itsm.srv.install.doc/b_srv_install_guide_linux.pdf, verfügbar am 16.08.2012

[INF2006] Rechenberger, Peter: Informatik Handbuch, Hanser Verlag 2006, ISBN 3446218424

[ISCSI2006] Hufferd, John L.: ISCSI The Universal Storage Connection, Boston, Pearson Education Inc. 2006, ISBN 0-201-78419-X

[KERN2012] Svec, Martin: Fix unsupported WRITE_SAME sense payload. URL: http://www.kernel.org/pub/linux/kernel/v3.x/ChangeLog-3.2.15, verfügbar am 10.10.2012

[NFS1991] Santifaller, Michael: TCP/IP und NFS in Theorie und Praxis, Addison-Wesley 1991, ISBN 3-89319-246-8

[OPENA2012] Silverberg, Mark: Administration Guide. URL: http://opendedup.org/administrators-guide, verfügbar am 23.08.2012

[OPENP2010] Silverberg, Mark: SDFS Architecture Presentation. URL: http://opendedup.googlecode.com/files/SDFS%20Architecture.pdf, verfügbar am 23.08.2012

[OPENG2012] Silverberg, Mark: OpenDedup Deduplication NAS Appliance Administration. URL: http://www.opendedup.org/nasapp, verfügbar am 23.08.2012

[QNA2012] QNAP: QNAP Turbo NAS Software User Manual. URL: http://docs.qnap.com/nas/en/index.html, verfügbar am 12.10.2012

[QUA2011] Mark R. Coppock: Data De-duplication for DUMMIES. URL: http://www.quantum.com/go/Deduplication.html, verfügbar am 10.10.2012

[RFC1014] SUN: XDR: External Data Representation Standard. URL: http://tools.ietf.org/html/rfc1014, verfügbar am 10.10.2012

[RFC1094] SUN: NFS: Network File System Protocol Specification. URL: http://tools.ietf.org/html/rfc1094, verfügbar am 10.10.2012

[SNIA2012] SNIA: Dictionary D.URL: http://www.snia.org/education/dictionary/d, verfügbar am 10.10.2012

[SPE2010] Walter, Christian: Daten-Deduplizierung als Replikationslösung im praktischen Einsatz. URL: http://www.speicherguide.de/backup-recovery/disk-backup/rheinwohnungsbau-erreicht-dedup-rate-von-112-12275.aspx, verfügbar am 16.08.2012

[SNI2008] Dutch, Mike: Understanding data deduplication ratios. URL: http://www.snia.org/sites/default/files/Understanding_Data_Deduplication_Ratios-20080718.pdf, verfügbar am 16.08.2012

[TAN2009] Tanenbaum, Andrew S.: Moderne Betriebssysteme. Pearson Education 2009, ISBN: 3-82737-019-1

[NFS1998] Stern, Hal: Verwaltung von UNIX-Netzwerken mit NFS und NIS, O'Reilly Verlag 1998, ISBN: 3-930673-24-8

[VMB2012] Hogan, Cormac: Low Level VAAI Behavior. URL: http://blogs.vmware.com/vsphere/2012/06/low-level-vaai-behaviour.html, verfügbar am 11.10.2012

[VMD2012] VMware. URL: http://pubs.vmware.com/vsphere-51/index.jsp?topic=%2Fcom.vmware.vsphere.install.doc%2FGUID-7C9A1E23-7FCD-4295-9CB1-C932F2423C63.html, verfügbar am 01.10.2012

[VMW2012] VMware: Disabling the VAAI functionality in ESX/ESXI. URL: http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1033665, verfügbar am 07.10.2012

[WH2012] Wikipedia: Hashfunktion. URL: http://de.wikipedia.org/wiki/Hashfunktion, verfügbar am 15.09.2012

[WQ2012] Wikipedia: Quersumme. URL: http://de.wikipedia.org/wiki/Quersumme, verfügbar am 15.09.2012

[WI2012] Wikipedia: ISBN. URL: http://de.wikipedia.org/wiki/ISBN, verfügbar am 15.09.2012

[WS2012] Wikipedia: Small Computer Interface URL: http://de.wikipedia.org/wiki/SCSI, verfügbar am 15.09.2012

[WIKI2012] Wikipedia: Server Message Block. URL: http://de.wikipedia.org/wiki/Server_Message_Block, verfügbar am 24.08.2012

Anlagen

Anlagen, Teil 1

Einen Überblick über die benutzte Verzeichnisstruktur der SDFS-Appliance zeigt die nachfolgende Darstellung.

```
Wurzel der LNX-Maschine
|
+---etc
|   \---sdfs
|          cfsdfs_4k_vol01-volume-cfg.xml
|          cfsdfs_4k_vol01-volume-cfg.xml
|          cfsdfs_4k_vol03-volume-cfg.xml
|          cfsdfs_4k_vol04-volume-cfg.xml
|          cfsdfs_4k_vol05-volume-cfg.xml
|          cfsdfs_4k_vol06-volume-cfg.xml
|          replication.example.props
|          sample-dse-cfg.xml
|          routing-config.xml
|
+---var
|    \---log
|          \---sdfs
|               sdfs.log
|               sdfs_viewer.log
|               cfsdfs_4k_vol01-volume-cfg.xml.log
|               cfsdfs_4k_vol02-volume-cfg.xml.log
|               cfsdfs_4k_vol03-volume-cfg.xml.log
|               cfsdfs_4k_vol04-volume-cfg.xml.log
|               cfsdfs_4k_vol05-volume-cfg.xml.log
|               cfsdfs_4k_vol06-volume-cfg.xml.log
+---media
|    \---sdfs
|          \--volumes
|               \---cfsdfs_4k_vol01
|               \---cfsdfs_4k_vol02
|               \---cfsdfs_4k_vol03
|               \---cfsdfs_4k_vol04
|               \---cfsdfs_4k_vol05
|               \---cfsdfs_4k_vol06
+---opt
|    \---sdfs
|          \---volumes
|               \---cfsdfs_4k_vol01
|                    \---chunkstore
|                    \---ddb
```

```
|                    \---files
|               \---cfsdfs_4k_vol02
|                    \---chunkstore
|                    \---ddb
|                    \---files
|
|               \---cfsdfs_4k_vol03
|                    \---chunkstore
|                    \---ddb
|                    \---files
|
|               \---cfsdfs_4k_vol04
|                    \---chunkstore
|                    \---ddb
|                    \---files
|
|               \---cfsdfs_4k_vol05
|                    \---chunkstore
|                    \---ddb
|                    \---files
|
|               \---cfsdfs_4k_vol06
|                    \---chunkstore
|                    \---ddb
|                    \---files
|
```

Anlagen, Teil 2

Bei dieser XML-Datei handelt es sich um die Volume-Konfigurationsdatei eines SDFS-Volumes, das für die DEDUP-Ratio und Performance-Überprüfungen verwendet wird.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<subsystem-config version="1.1.8">
<locations dedup-db-store="/opt/sdfs/volumes/cfsdfs_4k_vol01/ddb" io-
log="/opt/sdfs/volumes/cfsdfs_4k_vol01/ioperf.log"/>
<io chunk-size="4" claim-hash-schedule="0 0 0/2 * * ?" dedup-files="true"
file-read-cache="5" hash-type="tiger16" log-level="1" max-file-
inactive="900" max-file-write-buffers="1" max-open-files="1024" meta-
file-cache="1024" multi-read-timeout="1000" safe-close="false" safe-
sync="false" system-read-cache="1000" write-threads="12"/>
<permissions default-file="0644" default-folder="0755" default-group="0"
default-owner="0"/>

<launch-params class-path="/usr/share/sdfs/lib/snappy-
java.jar:/usr/share/sdfs/lib/activation-
1.1.jar:/usr/share/sdfs/lib/antlr-2.7.4.jar:/usr/share/sdfs/lib/apache-
mime4j-0.6.jar:/usr/share/sdfs/lib/bcprov-jdk16-
143.jar:/usr/share/sdfs/lib/chardet-1.0.jar:/usr/share/sdfs/lib/commons-
cli-1.2.jar:/usr/share/sdfs/lib/commons-codec-
1.3.jar:/usr/share/sdfs/lib/commons-collections-
3.2.1.jar:/usr/share/sdfs/lib/commons-digester-
1.8.1.jar:/usr/share/sdfs/lib/commons-httpclient-
3.1.jar:/usr/share/sdfs/lib/commons-io-
1.4.jar:/usr/share/sdfs/lib/commons-lang3-
3.1.jar:/usr/share/sdfs/lib/commons-logging-
1.0.4.jar:/usr/share/sdfs/lib/commons-logging-
1.1.1.jar:/usr/share/sdfs/lib/commons-pool-
1.5.5.jar:/usr/share/sdfs/lib/concurrent-
1.3.4.jar:/usr/share/sdfs/lib/concurrentlinkedhashmap-lru-
1.3.jar:/usr/share/sdfs/lib/cpdetector_1.0.8.jar:/usr/share/sdfs/lib/dom4
j-1.6.1.jar:/usr/share/sdfs/lib/httpclient-
4.1.1.jar:/usr/share/sdfs/lib/httpcore-
4.1.jar:/usr/share/sdfs/lib/httpcore-nio-
4.1.jar:/usr/share/sdfs/lib/httpmime-
4.0.3.jar:/usr/share/sdfs/lib/jackson-core-asl-
1.8.3.jar:/usr/share/sdfs/lib/jackson-jaxrs-
1.8.3.jar:/usr/share/sdfs/lib/jackson-mapper-asl-
1.8.3.jar:/usr/share/sdfs/lib/jackson-xc-
1.8.3.jar:/usr/share/sdfs/lib/jacksum.jar:/usr/share/sdfs/lib/jargs-
1.0.jar:/usr/share/sdfs/lib/javax.inject-1.jar:/usr/share/sdfs/lib/java-
xmlbuilder-1.jar:/usr/share/sdfs/lib/jaxb-impl-2.2.3-
1.jar:/usr/share/sdfs/lib/jaxen-1.1.1.jar:/usr/share/sdfs/lib/jcs-
1.3.jar:/usr/share/sdfs/lib/jdbm.jar:/usr/share/sdfs/lib/jdokan.jar:/usr/
share/sdfs/lib/jersey-client-1.10-b02.jar:/usr/share/sdfs/lib/jersey-
core-1.10-b02.jar:/usr/share/sdfs/lib/jersey-json-1.10-
b02.jar:/usr/share/sdfs/lib/jets3t-0.7.4.jar:/usr/share/sdfs/lib/jets3t-
0.8.1.jar:/usr/share/sdfs/lib/log4j-1.2.15.jar:/usr/share/sdfs/lib/mail-
1.4.jar:/usr/share/sdfs/lib/microsoft-windowsazure-api-
0.2.2.jar:/usr/share/sdfs/lib/quartz-
1.8.3.jar:/usr/share/sdfs/lib/sdfs.jar:/usr/share/sdfs/lib/simple-
4.1.21.jar:/usr/share/sdfs/lib/slf4j-api-
1.5.10.jar:/usr/share/sdfs/lib/slf4j-log4j12-
1.5.10.jar:/usr/share/sdfs/lib/stax-api-
```

```
1.0.1.jar:/usr/share/sdfs/lib/trove-
3.0.0a3.jar:/usr/share/sdfs/lib/truezip-samples-7.3.2-jar-with-
dependencies.jar:/usr/share/sdfs/lib/uuid-3.1.jar" java-options="-
Djava.library.path=/usr/share/sdfs/bin/ -
Dorg.apache.commons.logging.Log=fuse.logging.FuseLog -
Dfuse.logging.level=INFO -server -XX:+UseG1GC -Xmx1328m -Xmn328m" java-
path="/usr/share/sdfs/jre1.7.0/bin/java"/>
<sdfscli enable="true" enable-auth="false" listen-address="localhost"
pass-
word="7e7332f85e4123877f844298f89eaae326b42430b890f6483a62ec89ce3fb80f"
port="6442" salt="biIr9N"/>
<local-chunkstore allocation-size="107374182400" chunk-gc-schedule="0 0
0/4 * * ?" chunk-
store="/opt/sdfs/volumes/cfsdfs_4k_vol01/chunkstore/chunks" chunk-store-
dirty-timeout="1000" chunk-store-read-cache="5" chunkstore-
class="org.opendedup.sdfs.filestore.FileChunkStore" enabled="true" en-
crypt="false" encryption-key="J6Q7Sl9Mzs4a@HsCM7nBMGsjkJ2M-uqMhlz" evic-
tion-age="6" gc-class="org.opendedup.sdfs.filestore.gc.PFullGC" hash-db-
store="/opt/sdfs/volumes/cfsdfs_4k_vol01/chunkstore/hdb" max-repl-batch-
sz="128" pre-allocate="false" read-ahead-pages="8">
<network enable="false" hostname="0.0.0.0" port="2222" upstream-
enabled="false" upstream-host="" upstream-host-port="2222" upstream-
password="admin" use-udp="false"/>
</local-chunkstore>
<volume allow-external-links="true" capacity="500GB" closed-
gracefully="false" current-size="519880979312" duplicate-
bytes="27490750464" maximum-percentage-full="-1.0"
path="/opt/sdfs/volumes/cfsdfs_4k_vol01/files" read-bytes="10604634868"
write-bytes="11726372864"/>
</subsystem-config>
```

Anlagen, Teil 3

Die Installationseinstellungen der virtuellen Maschinen sind in dieser Konfigurationsdatei gelistet.

```
# Kickstart file automatically generated by anaconda.

#version=DEVEL
install
cdrom
lang en_US.UTF-8
keyboard us
network --onboot no --device eth0 --bootproto dhcp --noipv6
rootpw  --iscrypted
$6$ZOWGmxdowCslNxEC$IapUNGWohQzxLnRWcwMVvMkBYDLv7Dm8qLDqJlsKjCSH6VJjJ99XB
PJKIsXjvtvWICNEyftWoPMU0e6lyM9rj0
firewall --service=ssh
authconfig --enableshadow --passalgo=sha512
selinux --enforcing
timezone --utc America/New_York
bootloader --location=mbr --driveorder=sda --append="crashkernel=auto
rhgb quiet"
# The following is the partition information you requested
# Note that any partitions you deleted are not expressed
# here so unless you clear all partitions first, this is
# not guaranteed to work
#clearpart --all --drives=sda

#part /boot --fstype=ext4 --size=500
#part pv.008002 --grow --size=1

#volgroup vg_cfvirt01 --pesize=4096 pv.008002
#logvol /home --fstype=ext4 --name=lv_home --vgname=vg_cfvirt01 --grow --
size=100
#logvol / --fstype=ext4 --name=lv_root --vgname=vg_cfvirt01 --grow --
size=1024 --maxsize=51200
#logvol swap --name=lv_swap --vgname=vg_cfvirt01 --grow --size=4032 --
maxsize=4032


%packages
@base
@client-mgmt-tools
@core
@debugging
@basic-desktop
@desktop-debugging
@desktop-platform
@directory-client
@fonts
@general-desktop
@graphical-admin-tools
@input-methods
@internet-applications
@internet-browser
@java-platform
```

```
@kde-desktop
@legacy-x
@network-file-system-client
@office-suite
@print-client
@remote-desktop-clients
@server-platform
@workstation-policy
@x11
pax
python-dmidecode
oddjob
wodim
sgpio
genisoimage
mtools
abrt-gui
qt-mysql
certmonger
pam_krb5
krb5-workstation
gnome-pilot
xterm
xorg-x11-xdm
libXmu
rdesktop
%end
```

Anlagen, Teil 4

Beispielhafte ESX-Konfigurationsdatei der eingesetzten virtuellen Maschinen.

```
cfvirt01 - RH6 - Standard-Installation.vmx
.encoding = "UTF-8"
config.version = "8"
virtualHW.version = "8"
pciBridge0.present = "true"
pciBridge4.present = "true"
pciBridge4.virtualDev = "pcieRootPort"
pciBridge4.functions = "8"
pciBridge5.present = "true"
pciBridge5.virtualDev = "pcieRootPort"
pciBridge5.functions = "8"
pciBridge6.present = "true"
pciBridge6.virtualDev = "pcieRootPort"
pciBridge6.functions = "8"
pciBridge7.present = "true"
pciBridge7.virtualDev = "pcieRootPort"
pciBridge7.functions = "8"
vmci0.present = "true"
hpet0.present = "true"
nvram = "cfvirt01 - RH6 - Standard-Installation.nvram"
virtualHW.productCompatibility = "hosted"
powerType.powerOff = "soft"
powerType.powerOn = "hard"
powerType.suspend = "hard"
powerType.reset = "soft"
displayName = "cfvirt01 - RH6 - Standard-Installation"
extendedConfigFile = "cfvirt01 - RH6 - Standard-Installation.vmxf"
floppy0.present = "true"
scsi0.present = "true"
scsi0.sharedBus = "none"
scsi0.virtualDev = "pvscsi"
memsize = "2048"
scsi0:0.present = "true"
scsi0:0.fileName = "cfvirt01 - RH6 - Standard-Installation.vmdk"
scsi0:0.deviceType = "scsi-hardDisk"
sched.scsi0:0.shares = "normal"
sched.scsi0:0.throughputCap = "off"
ide1:0.present = "true"
ide1:0.clientDevice = "true"
ide1:0.deviceType = "atapi-cdrom"
ide1:0.startConnected = "false"
floppy0.startConnected = "false"
floppy0.fileName = ""
floppy0.clientDevice = "true"
ethernet0.present = "true"
ethernet0.virtualDev = "vmxnet3"
ethernet0.networkName = "VM Network"
ethernet0.addressType = "vpx"
ethernet0.generatedAddress = "00:50:56:be:80:0a"
svga.vramSize = "8388608"
guestOS = "rhel6-64"
uuid.bios = "42 3e d3 d4 e8 4c 0b e3-2a 86 b6 bd 40 7a c7 bd"
```

```
vc.uuid = "50 3e bf bc 78 14 4d 2c-35 2a c7 af 4f b0 1d 5c"
snapshot.action = "keep"
sched.cpu.min = "0"
sched.cpu.units = "mhz"
sched.cpu.shares = "normal"
sched.mem.min = "0"
sched.mem.shares = "normal"
tools.upgrade.policy = "manual"
ethernet0.pciSlotNumber = "192"
evcCompatibilityMode = "FALSE"
guestCPUID.0 = "0000000d756e65476c65746e49656e69"
guestCPUID.1 = "0001067a00010800840822010febfbff"
guestCPUID.80000001 = "00000000000000000000000120100800"
hostCPUID.0 = "0000000d756e65476c65746e49656e69"
hostCPUID.1 = "0001067a000208000408e3fdbfebfbff"
hostCPUID.80000001 = "00000000000000000000000120100800"
pciBridge0.pciSlotNumber = "17"
pciBridge4.pciSlotNumber = "21"
pciBridge5.pciSlotNumber = "22"
pciBridge6.pciSlotNumber = "23"
pciBridge7.pciSlotNumber = "24"
replay.filename = ""
replay.supported = "TRUE"
scsi0.pciSlotNumber = "160"
scsi0.sasWWID = "50 05 05 64 e8 4c 0b e0"
scsi0:0.redo = ""
userCPUID.0 = "0000000d756e65476c65746e49656e69"
userCPUID.1 = "0001067a00020800040822010febfbff"
userCPUID.80000001 = "00000000000000000000000120100800"
vmci0.pciSlotNumber = "32"
vmotion.checkpointFBSize = "8388608"
tools.remindInstall = "TRUE"
vmci0.id = "1081788349"
uuid.location = "56 4d 1f 1d ce 32 0c d9-63 58 45 90 ad 2b 5f 38"
cleanShutdown = "TRUE"
sched.swap.derivedName = "/vmfs/volumes/506b4dcf-7dd7ea18-2736-
90e2ba010705/cfvirt01 - RH6 - Standard-Installation/cfvirt01 - RH6 -
Standard-Installation-e82355bd.vswp"
```

Anlagen, Teil 5

Installations-Konfigurationsdatei der eingesetzten virtuellen Maschinen, in Abschnitt 4.5.3.

```
# Kickstart file automatically generated by anaconda.

#version=DEVEL
install
cdrom
lang en_US.UTF-8
keyboard de
network --onboot no --device eth0 --bootproto dhcp --noipv6
rootpw  --iscrypted
$6$2dhVgJHHbyfgTQx8$N2xxJtPfCIs4pLpPVFo/EqRIGKbVa7o4uGR8W29XeRJdtPdcYCgCG
LF6izNH3suq7KpCoXj4Gjl9UWF9mhd4L.
firewall --service=ssh
authconfig --enableshadow --passalgo=sha512
selinux --enforcing
timezone --utc Europe/Vienna
bootloader --location=mbr --driveorder=sda --append="crashkernel=auto
rhgb quiet"
# The following is the partition information you requested
# Note that any partitions you deleted are not expressed
# here so unless you clear all partitions first, this is
# not guaranteed to work
#clearpart --all --drives=sda

#part /boot --fstype=ext4 --size=500
#part pv.008002 --grow --size=1

#volgroup vg_cfvirt25 --pesize=4096 pv.008002
#logvol / --fstype=ext4 --name=lv_root --vgname=vg_cfvirt25 --size=232968
#logvol swap --name=lv_swap --vgname=vg_cfvirt25 --size=2048


%packages
@base
@client-mgmt-tools
@core
@debugging
@basic-desktop
@desktop-debugging
@desktop-platform
@directory-client
@fonts
@general-desktop
@graphical-admin-tools
@input-methods
@internet-applications
@internet-browser
@java-platform
@kde-desktop
@legacy-x
@network-file-system-client
@office-suite
@print-client
```

```
@remote-desktop-clients
@server-platform
@workstation-policy
@x11
pax
python-dmidecode
oddjob
wodim
sgpio
genisoimage
mtools
abrt-gui
qt-mysql
certmonger
pam_krb5
krb5-workstation
gnome-pilot
xterm
xorg-x11-xdm
libXmu
rdesktop
%end
```

Anlagen, Teil 6

Konfigurationsdatei des eingesetzten SDFS-Volumes in Abschnitt 4.5.3.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><subsystem-config
version="1.1.8">
<locations dedup-db-store="/opt/sdfs/volumes/cfsdfs_4k_vol09/ddb" io-
log="/opt/sdfs/volumes/cfsdfs_4k_vol09/ioperf.log"/>
<io chunk-size="4" claim-hash-schedule="0 0 0/2 * * ?" dedup-files="true"
file-read-cache="5" hash-type="tiger16" log-level="1" max-file-
inactive="900" max-file-write-buffers="1" max-open-files="1024" meta-
file-cache="1024" multi-read-timeout="1000" safe-close="false" safe-
sync="false" system-read-cache="1000" write-threads="12"/>
<permissions default-file="0644" default-folder="0755" default-group="0"
default-owner="0"/>


<launch-params class-path="/usr/share/sdfs/lib/snappy-
java.jar:/usr/share/sdfs/lib/activation-
1.1.jar:/usr/share/sdfs/lib/antlr-2.7.4.jar:/usr/share/sdfs/lib/apache-
mime4j-0.6.jar:/usr/share/sdfs/lib/bcprov-jdk16-
143.jar:/usr/share/sdfs/lib/chardet-1.0.jar:/usr/share/sdfs/lib/commons-
cli-1.2.jar:/usr/share/sdfs/lib/commons-codec-
1.3.jar:/usr/share/sdfs/lib/commons-collections-
3.2.1.jar:/usr/share/sdfs/lib/commons-digester-
1.8.1.jar:/usr/share/sdfs/lib/commons-httpclient-
3.1.jar:/usr/share/sdfs/lib/commons-io-
1.4.jar:/usr/share/sdfs/lib/commons-lang3-
3.1.jar:/usr/share/sdfs/lib/commons-logging-
1.0.4.jar:/usr/share/sdfs/lib/commons-logging-
1.1.1.jar:/usr/share/sdfs/lib/commons-pool-
1.5.5.jar:/usr/share/sdfs/lib/concurrent-
1.3.4.jar:/usr/share/sdfs/lib/concurrentlinkedhashmap-lru-
1.3.jar:/usr/share/sdfs/lib/cpdetector_1.0.8.jar:/usr/share/sdfs/lib/dom4
j-1.6.1.jar:/usr/share/sdfs/lib/httpclient-
4.1.1.jar:/usr/share/sdfs/lib/httpcore-
4.1.jar:/usr/share/sdfs/lib/httpcore-nio-
4.1.jar:/usr/share/sdfs/lib/httpmime-
4.0.3.jar:/usr/share/sdfs/lib/jackson-core-asl-
1.8.3.jar:/usr/share/sdfs/lib/jackson-jaxrs-
1.8.3.jar:/usr/share/sdfs/lib/jackson-mapper-asl-
1.8.3.jar:/usr/share/sdfs/lib/jackson-xc-
1.8.3.jar:/usr/share/sdfs/lib/jacksum.jar:/usr/share/sdfs/lib/jargs-
1.0.jar:/usr/share/sdfs/lib/javax.inject-1.jar:/usr/share/sdfs/lib/java-
xmlbuilder-1.jar:/usr/share/sdfs/lib/jaxb-impl-2.2.3-
1.jar:/usr/share/sdfs/lib/jaxen-1.1.1.jar:/usr/share/sdfs/lib/jcs-
1.3.jar:/usr/share/sdfs/lib/jdbm.jar:/usr/share/sdfs/lib/jdokan.jar:/usr/
share/sdfs/lib/jersey-client-1.10-b02.jar:/usr/share/sdfs/lib/jersey-
core-1.10-b02.jar:/usr/share/sdfs/lib/jersey-json-1.10-
b02.jar:/usr/share/sdfs/lib/jets3t-0.7.4.jar:/usr/share/sdfs/lib/jets3t-
0.8.1.jar:/usr/share/sdfs/lib/log4j-1.2.15.jar:/usr/share/sdfs/lib/mail-
1.4.jar:/usr/share/sdfs/lib/microsoft-windowsazure-api-
0.2.2.jar:/usr/share/sdfs/lib/quartz-
1.8.3.jar:/usr/share/sdfs/lib/sdfs.jar:/usr/share/sdfs/lib/simple-
4.1.21.jar:/usr/share/sdfs/lib/slf4j-api-
1.5.10.jar:/usr/share/sdfs/lib/slf4j-log4j12-
```

```
1.5.10.jar:/usr/share/sdfs/lib/stax-api-
1.0.1.jar:/usr/share/sdfs/lib/trove-
3.0.0a3.jar:/usr/share/sdfs/lib/truezip-samples-7.3.2-jar-with-
dependencies.jar:/usr/share/sdfs/lib/uuid-3.1.jar" java-options="-
Djava.library.path=/usr/share/sdfs/bin/ -
Dorg.apache.commons.logging.Log=fuse.logging.FuseLog -
Dfuse.logging.level=INFO -server -XX:+UseG1GC -Xmx2528m -Xmn528m" java-
path="/usr/share/sdfs/jre1.7.0/bin/java"/>
<sdfscli enable="true" enable-auth="false" listen-address="localhost"
pass-
word="0ec6db9eff6235035943ddaeaf9b596a8ab737242f52881945172d97413ee9ff"
port="6442" salt="Mfl1J0"/>
<local-chunkstore allocation-size="268435456000" chunk-gc-schedule="0 0
0/4 * * ?" chunk-
store="/opt/sdfs/volumes/cfsdfs_4k_vol09/chunkstore/chunks" chunk-store-
dirty-timeout="1000" chunk-store-read-cache="5" chunkstore-
class="org.opendedup.sdfs.filestore.FileChunkStore" enabled="true" en-
crypt="false" encryption-key="SVjk26Qbj4BWEP12rGJwdBUdNNlWFteRhvJ" evic-
tion-age="6" gc-class="org.opendedup.sdfs.filestore.gc.PFullGC" hash-db-
store="/opt/sdfs/volumes/cfsdfs_4k_vol09/chunkstore/hdb" max-repl-batch-
sz="128" pre-allocate="false" read-ahead-pages="8">
<network enable="false" hostname="0.0.0.0" port="2222" upstream-
enabled="false" upstream-host="" upstream-host-port="2222" upstream-
password="admin" use-udp="false"/>
</local-chunkstore>
<volume allow-external-links="true" capacity="250GB" closed-
gracefully="false" current-size="268435456512" duplicate-
bytes="6906138624" maximum-percentage-full="-1.0"
path="/opt/sdfs/volumes/cfsdfs_4k_vol09/files" read-bytes="35313123328"
write-bytes="213882368000"/></subsystem-config>
```

Anlagen, Teil 7

Eingesetztes BASH-Script für die Zeitmessung in Abschnitt 4.5.3

```
#!/bin/sh

#configuration
source="/source"
target="/tmp/target"
log="./log_29.10.2012_cfvirt26_zero.txt"

#sync filesystem and drop caches
sync
echo 3 > /proc/sys/vm/drop_caches
echo "" > ${log}


#script start
if [ ! -d "${target}" ]; then
mkdir -p ${target}
fi
COUNTER=1
for i in $(ls ${source}/*.dd)
do
  echo 3 > /proc/sys/vm/drop_caches
  echo "copy ${i} from ${source} to ${target}" >> ${log}
  echo "START ${COUNTER}: `date +%H:%M:%S`" >> ${log}
  { time (cp $i $target/ && sync) } 2>>  ${log}
  echo "END ${COUNTER}: `date +%H:%M:%S`" >> ${log}
  let COUNTER=COUNTER+1
done
```

Anlagen, Teil 8

Belegter Speicherplatz innerhalb der SDFS-Appliance nach der ersten Standard-Installation einer virtuellen Maschine.

```
Thu Oct  4 21:37:35 CEST 2012
## list directory
total 128
drwxr-xr-x 1 root root 8388608 Oct  4 21:37 cfvirt01 - RH6 - Standard-Installation
## filesystem status
Filesystem                                          Size  Used Avail Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442  500G  101G  400G  21% /media/cfsdfs_4k_vol01
Filesystem                                         1K-blocks      Used Available Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442 524288000 104857728 419430272  21% /media/cfsdfs_4k_vol01
## sdfs-Information
Volume Capacity : 500 GB
Volume Current Size : 100 GB
Volume Max Percentage Full : Unlimited
Volume Duplicate Data Written : 2 GB
Volume Unique Data Written: 3.7 GB
Volume Data Read : 620.8 MB
Volume Virtual Dedup Rate (Dup/Total Bytes Written) : 35.58%
Volume Real Dedup Rate (DSE Size/Total Bytes Written) : 19.45%
Volume Actual Storage Savings (Unique Blocks Stored/Current Size) : 95.36%
## usage duplicated
3891164 /media/cfsdfs_4k_vol01/
3.8G    /media/cfsdfs_4k_vol01/
## usage deduplicated
4987788 /opt/sdfs/volumes/cfsdfs_4k_vol01/
4.8G    /opt/sdfs/volumes/cfsdfs_4k_vol01/
```

Belegter Speicherplatz innerhalb der SDFS-Appliance nach der zweiten Standard-Installation einer virtuellen Maschine.

```
Thu Oct  4 22:00:13 CEST 2012
## list directory
total 256
drwxr-xr-x 1 root root 8388608 Oct  4 21:37 cfvirt01 - RH6 - Standard-Installation
drwxr-xr-x 1 root root 8388608 Oct  4 21:58 cfvirt02 - RH6 - Standard-Installation
## filesystem status
Filesystem                                          Size  Used Avail Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442  500G  201G  300G  41% /media/cfsdfs_4k_vol01
Filesystem                                         1K-blocks      Used Available Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442 524288000 209715456 314572544  41% /media/cfsdfs_4k_vol01
## sdfs-Information
Volume Capacity : 500 GB
Volume Current Size : 200 GB
Volume Max Percentage Full : Unlimited
Volume Duplicate Data Written : 7.3 GB
Volume Unique Data Written: 4.4 GB
Volume Data Read : 1 GB
Volume Virtual Dedup Rate (Dup/Total Bytes Written) : 62.43%
Volume Real Dedup Rate (DSE Size/Total Bytes Written) : 56.92%
Volume Actual Storage Savings (Unique Blocks Stored/Current Size) : 97.48%
## usage duplicated
4605480 /media/cfsdfs_4k_vol01/
4.4G    /media/cfsdfs_4k_vol01/
## usage deduplicated
5449272 /opt/sdfs/volumes/cfsdfs_4k_vol01/
5.2G    /opt/sdfs/volumes/cfsdfs_4k_vol01/
```

Belegter Speicherplatz innerhalb der SDFS-Appliance nach der dritten Standard-Installation einer virtuellen Maschine.

```
Thu Oct  4 22:33:21 CEST 2012
## list directory
total 384
drwxr-xr-x 1 root root 8388608 Oct  4 21:37 cfvirt01 - RH6 - Standard-Installation
drwxr-xr-x 1 root root 8388608 Oct  4 21:58 cfvirt02 - RH6 - Standard-Installation
drwxr-xr-x 1 root root 8388608 Oct  4 22:32 cfvirt03 - RH6 - Standard-Installation
## filesystem status
Filesystem                                          Size  Used Avail Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442  500G  301G  200G  61% /media/cfsdfs_4k_vol01
Filesystem                                         1K-blocks      Used Available Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442 524288000 314573184 209714816  61% /media/cfsdfs_4k_vol01
## sdfs-Information
Volume Capacity : 500 GB
Volume Current Size : 300 GB
Volume Max Percentage Full : Unlimited
Volume Duplicate Data Written : 12.6 GB
Volume Unique Data Written: 5 GB
Volume Data Read : 1.5 GB
Volume Virtual Dedup Rate (Dup/Total Bytes Written) : 71.45%
Volume Real Dedup Rate (DSE Size/Total Bytes Written) : 69.29%
Volume Actual Storage Savings (Unique Blocks Stored/Current Size) : 98.19%
## usage duplicated
5291436 /media/cfsdfs_4k_vol01/
5.1G    /media/cfsdfs_4k_vol01/
## usage deduplicated
5904096 /opt/sdfs/volumes/cfsdfs_4k_vol01/
5.7G    /opt/sdfs/volumes/cfsdfs_4k_vol01/
```

Belegter Speicherplatz innerhalb der SDFS-Appliance nach der vierten Standard-Installation einer virtuellen Maschine.

```
Thu Oct  4 23:08:07 CEST 2012
## list directory
total 512
drwxr-xr-x 1 root root 8388608 Oct  4 21:37 cfvirt01 - RH6 - Standard-Installation
drwxr-xr-x 1 root root 8388608 Oct  4 21:58 cfvirt02 - RH6 - Standard-Installation
drwxr-xr-x 1 root root 8388608 Oct  4 22:32 cfvirt03 - RH6 - Standard-Installation
drwxr-xr-x 1 root root 8388608 Oct  4 23:01 cfvirt04 - RH6 - Standard-Installation
## filesystem status
Filesystem                                          Size  Used Avail Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442  500G  401G  100G  81% /media/cfsdfs_4k_vol01
Filesystem                                         1K-blocks      Used Available Use% Mounted on
sdfs:/etc/sdfs/cfsdfs_4k_vol01-volume-cfg.xml:6442 524288000 419431040 104856960  81% /media/cfsdfs_4k_vol01
## sdfs-Information
Volume Capacity : 500 GB
Volume Current Size : 400 GB
Volume Max Percentage Full : Unlimited
Volume Duplicate Data Written : 17.9 GB
Volume Unique Data Written: 5.8 GB
Volume Data Read : 1.9 GB
Volume Virtual Dedup Rate (Dup/Total Bytes Written) : 75.67%
Volume Real Dedup Rate (DSE Size/Total Bytes Written) : 75.35%
Volume Actual Storage Savings (Unique Blocks Stored/Current Size) : 98.54%
## usage duplicated
6031688 /media/cfsdfs_4k_vol01/
5.8G    /media/cfsdfs_4k_vol01/
## usage deduplicated
6367892 /opt/sdfs/volumes/cfsdfs_4k_vol01/
6.1G    /opt/sdfs/volumes/cfsdfs_4k_vol01/
```

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Weinitzen, den 02.11.2012

Christian Neuhold